UCI-AM-16-016

ACQUISITION RESEARCH PROGRAM
SPONSORED REPORT SERIES

Achieving Better Buying Power through Acquisition of Open Architecture Software Systems for Web and Mobile Devices

22 February 2016

Dr. Walt Scacchi

Dr. Thomas A. Alspaugh

Institute for Software Research University

University of California, Irvine

Approved for public release; distribution is unlimited.

Prepared for the Naval Postgraduate School, Monterey, CA 93943.

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC
POLICY NAVAL POSTGRADUATE SCHOOL

The research presented in this report was supported by the Acquisition Research Program of the Graduate School of Business & Public Policy at the Naval Postgraduate School.

To request defense acquisition research, to become a research sponsor, or to print additional copies of reports, please contact any of the staff listed on the Acquisition Research Program website (www.acquisitionresearch.net).

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

About the Authors

Dr. Walt Scacchi is a senior research scientist and research faculty member at the Institute for Software Research, University of California, Irvine. He received a PhD in information and computer science from U.C. Irvine in 1981. From 1981 to 1998 he was on the faculty at the University of Southern California. In 1999 he joined the Institute for Software Research at U.C. Irvine. He has published more than 150 research papers and has directed 60 externally funded research projects. In 2011 he served as co-chair for the 33rd International Conference on Software Engineering - Practice Track, and in 2012 he served as general co-chair of the 8th IFIP International Conference on Open Source Systems.

Dr. Walt Scacchi, Senior Research Scientist
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
E-mail: wscacchi@ics.uci.edu

Dr. Thomas Alspaugh is a project scientist at the Institute for Software Research, University of California, Irvine. His research interests are in software engineering, requirements, and licensing. Before completing his PhD, he worked as a software developer, team lead, and manager in industry, and as a computer scientist at the Naval Research Laboratory on the Software Cost Reduction or A-7 project.

Dr. Thomas Alspaugh, Project Scientist
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
E-mail: thomas.alspaugh@acm.org

THIS PAGE LEFT INTENTIONALLY BLANK

UCI-AM-16-016

ACQUISITION RESEARCH PROGRAM SPONSORED REPORT SERIES

Achieving Better Buying Power through Acquisition of Open Architecture Software Systems for Web and Mobile Devices

22 February 2016

Dr. Walt Scacchi

Dr. Thomas A. Alspaugh

Institute for Software Research University

University of California, Irvine

THIS PAGE LEFT INTENTIONALLY BLANK

Executive Summary

Many people within large enterprises rely on up to four Web-based or mobile devices for their daily work routines—personal computer, tablet, personal and work-specific smartphones. Our research was directed at identifying, tracking, and analyzing software component costs and cost reduction opportunities within acquisition life cycle of open architecture (OA) systems for such Web-based and mobile devices. These systems are subject to different intellectual property license and cybersecurity requirements. Our research goal was to create a new approach to address challenges in the acquisition of software systems for Web-based or mobile devices used within academic, business, or government enterprises. Acquisition personnel in such enterprises will increasingly be called on to review and approve choices between functionally similar open source software (OSS) components, and commercially priced closed source software (CSS) components, to be used in the design, implementation, deployment, and evolution of secure OA systems. Through our research, we sought to make this a simpler, more transparent, and more tractable process. Finally, this acquisition research supports and advances a public purpose by investigating acquisition challenges arising from the adoption and deployment of secure OA software systems for Web-based or mobile devices, which is of contemporary concern to most academic, business, or government enterprises.

Our research objective was to develop new ways and means for *identifying, tracking, and analyzing the costs and other better buying opportunities associated with the acquisition life cycle of OA software systems for Web-based or mobile devices.* OA system software elements can include either open source software (OSS) or proprietary closed-source software (CSS) components subject to different IP licenses and cybersecurity constraints. Such components may be configured into different, functionally similar versions that allow for common but costly CSS components to be replaced by their OSS counterparts, as a strategy to reduce software acquisition costs. Such replacement or substitution may arise at different stages of system acquisition including system design, integration, deployment, and evolution. But it is unclear what happens when the OA software components are widgets, apps, or mashups that arise from multi-party engineering efforts in heterogeneous software producer ecosystems, where end-users (or their home enterprise) are expected to serve as system integrators.

Web-based OA systems and mobile devices systems often integrate components independently developed by software producers using OSS or CSS, which then may be integrated into complete systems by system integrators [CoR14, GMO14, ReB12, RNC14, ScA14]**.** Acquisition personnel will increasingly be called on to review and approve security measures employed during the design, integration, deployment, and evolution of OA systems [ScA13b, ScA13c]. Our effort builds on related acquisition research efforts at the Software Engineering Institute (SEI) that address software product lines (SPLs) [BeJ10, JoB11], as well as on acquisition and development of secure OA systems built with widgets and apps [CBD14, CoR14, GGM14, GMH13, GMO14, ReB12, ReN14, ScA14, ScA14c]. Other related research addressing OSS [HiW10, Ke12, MarL11], component-based software ecosystems [End13, ReB12, RNC14, ScA12b, ScA13c, ScA14a, ScA14b], and better buying initiatives [ScA14]

informs us as well.

Our recent research demonstrates how complex OA systems can be designed, built, and deployed with alternative components and connectors resulting in functionally similar system versions, to satisfy overall system security requirements and individual system component intellectual property (IP) and cybersecurity requirements [ScA13a, ScA13b, ScA13c], as well as surfacing new challenges for achieving better buying power that can decrease (or increase) software acquisition costs [ScA14a, ScA14b, ScA15a]. Our next step is to identify, track, and analyze software acquisition and development practices associated with different types of Web-based and mobile software components including widgets, apps, and mashups. This may then help us to highlight opportunities to realize cost reduction and improve opportunities to realize better buying power. Our research results are applicable to most academic, business, or government enterprises that deploy complex information systems.

Our research results have been well received in presentations to different audiences, including within the larger Defense community, and the Federal Government more broadly. In particular, throughout 2015 our research results have been disseminated and picked up for use within the *Assembled Capabilities Working Group* (ACWG), under the guidance of the C3CB (Command, Control, Communications, and Business Systems) office within the DASD(A). This effort was facilitated through collaboration with many people from The MITRE Corporation, who along with the C3CB office are working in support of the Defense Intelligence Information Enterprise (DI2E). More broadly, our research results have been (or will be) presented to audiences at the 2015 Acquisition Research Symposium (Monterey, CA May 2015), the 2015 IEEE Software Technology Conference (Los Angeles, CA, 13 October 2015), and also as an invited tutorial at the 2016 Ground Systems Architecture Workshop (Los Angeles, CA, 29 February 2016, hosted by The Aerospace Corporation). Finally, one research paper has been submitted for review and publication, described below.

Finally, our research results are documented in chapters that follow in this Final Report. Chapter 1 is an overview of our research. Chapter 2 denotes our paper presented at the 2015 Acquisition Research Symposium, and also presented at the 2015 IEEE Software Technology Conference. Chapter 3 is our paper submitted for review and presentation at the 2016 Software Engineering and Industrial Practice Workshop at the International Conference on Software Engineering (Austin, TX, May 2016). Chapters 4 and 5 represent technical notes that were prepared, circulated, reviewed and discussed within the ACWG in support of the DASD(A)-C3CB office during Summer 2015. Finally, Chapter 6 provides a copy of an invited half-day tutorial at the 2016 Ground Systems Architecture Workshop for the Defense and Aerospace community.

Table of Contents

THIS PAGE LEFT INTENTIONALLY BLANK

Chapter 1:

Research Overview

Introduction

This research focuses on continuing investigation and refinement of techniques for identifying and reducing the costs, streamlining the process, and improving the readiness of future workforce for the acquisition of complex software systems. Emphasis is directed at identifying, tracking, and analyzing software component costs and cost reduction opportunities within acquisition life cycle of open architecture (OA) systems for Web-based and mobile devices, where such systems combine best-of-breed software components and software products lines (SPLs) that are subject to different intellectual property (IP) license and cybersecurity requirements.

This chapter provides an overview of the research effort during the period of 16 January 2015 through 15 January 2016. It includes a statement of work and description of the four research activities engaged during this period, followed by identification of the two acquisition research problems being investigated, the research and development basis for our research, and identification of our research publications that contain our studies and results. Each section is presented in turn.

Statement of Work and Research Description

Our objective was to develop new ways and means for identifying, tracking, and analyzing the costs associated with the acquisition life cycle of OA software systems. OA systems are those whose software elements can include either OSS or proprietary CSS components, where components are subject to different IP licenses and security constraints. Such components may be configured into different, functionally similar versions that allow for common but costly CSS components to be replaced by their OSS counterparts, as a strategy to reduce software acquisition costs. Such replacement or substitution may arise at different stages of system acquisition including system design, integration, deployment, and evolution. Recent DoD policy encourages the move to component-based OA software systems [DISA12, DISA12a, DoDOSA11], especially as DoD moves to embrace new mobile computing devices like smartphones and cloud-based software application services [DISA12a, Tak12].

Better Buying Power (BBP) is part of DoD's mandate to do more without more by implementing best practices in acquisition [Ken15]. BBP (up through 2013) identifies seven areas of focus organizing 36 initiatives that offer the potential to restore affordability in defense procurement and improve defense industry productivity. One area focuses on promoting competition, and includes an initiative to enforce open system architectures and effectively manage technical data rights.. Technical data rights pertain to two categories of IP: the Government's rights to (a) *technical data* (TD – product design data, computer databases, computer software documentation, etc.); and (b) *computer software* (CS – source code, executable code, processes, and related materials). These rights are controlled by IP licenses from system product or service providers (i.e. software producers) to the Government customer, imposing obligations the customer must fulfill (e.g., a fee paid in exchange for a certain number of software users authorized for the licensed product or service) [An12]. Our acquisition research has focused on issues addressing OA systems and IP licenses since 2008 [ScA08], and cybersecurity requirements since 2011 [ScA11].

OA software systems, integrated from components independently developed by different producers, offer the potential to reduce acquisition costs through new ways and means to acquire, develop, deploy, and sustain software-intensive systems. This may transform how DoD acquires complex systems by moving away from long-duration, proprietary (closed) system architectures with development costs that are difficult to control, towards more rapidly assembled/integrated OA systems with more transparent costs [ReB12, ScA13a, ScA13c]. Such a transformation may in turn reduce vendor lock-ins for deployed systems, often associated with rising costs and systems that are inaccessible to competing vendors. Our research on OA systems dating many years back [ScA08] has consistently been aligned with efforts to improve competition in software system development and evolution, through investigation of innovative ways and means to acquire/develop component-based OA software systems subject to diverse, heterogeneous IP licenses [AlS10]. But there is more to do to improve competition and defense affordability while effectively managing technical data rights in the acquisition of secure OA systems. There is a need to better understand how processes for acquiring cost-sensitive software systems are facilitated or constrained in light of overall BBP guidance and best practices, as well as how best to improve and streamline these processes when component-based OA software systems are being acquired.

We have sought to identify ways and means for streamlining the acquisition process for secure OA systems that incorporate Web-based or mobile devices through new ways and means for identifying, tracking, and analyzing OA software component costs. Such systems often integrate components independently developed by different software producers as either OSS or proprietary CSS. Program managers, acquisition officers, and contract managers will increasingly be called on to review and approve security measures employed during the design, integration, deployment, and evolution of OA systems [DoDOSA11, ScA13b, ScA13c]. Our effort builds on both our prior acquisition research [e.g., ScA08, ScA11, ScA12b, ScA13a, ScA14a] and related acquisition research efforts at the PEO IWS [GuC10, GuW12, WoS11], Department of the Navy [MaS12, GSS15], and Software Engineering Institute (SEI) that address SPLs [BeJ10, JoB11]. It is also influenced by related research in the DoD community addressing OSS [DISA12, HiW10, Ke12, MarL11], component-based software ecosystems [ReB12, ScA12b, ScA13c], and BBP initiatives [Ken15].

Realizing the objective of our investigation focused on *four project work activities*:

- Investigate the interactions between software system acquisition processes, and the cost consequences of alternative software system architectures incorporating different mixes of OSS and CSS widgets, apps, and mashup components subject to different licenses [ScA12b, ScA13a, ScA13b, ScA13c]. This entails exploring the balance between development, verification, and validation of software licenses and security rights, as well as the software widget, app, and mashup component/license costs while managing the development and evolution of OA systems at design-time, build-time, release and run-time.
- Developing formal foundations for establishing acquisition guidelines program

managers can use in reduced cost acquisition of software-intensive systems that rely on development and deployment of secure OA systems using OSS widget, app, and mashup technology and processes [ScA11, ScA12a, ScA12b, ScA13a, ScA13b, ScA13c, ScA14a, ScA14b, ScA14c].

- Continuing to develop concepts contributing to the emerging design of an automated approach supporting acquisition of secure, component-based, and increasingly mobile OA systems by (a) determining their conformance to acquisition guidelines/policies, contracts, and related license management issues, and (b) giving future acquisition workforce support and insights to properly review, approve, and manage the acquisition of complex systems that incorporate cost-sensitive acquisition of OA systems and software widget and application ("app") components [ScA11, ScA12a, ScA12b, ScA13a, ScA13b, ScA13c, ScA14a, ScA14b, ScA14c].
- Documenting the investigation, foundations, and results of the research in: (a) a Final Report delivered within 30 days of project completion to the Technical Point of Contact at NPS; (b) a research paper to be presented at the *12th Annual Acquisition Research Conference* in Monterey, CA, May 2015; (c) a progress report with the OSD sponsor via a video teleconference or other meeting at a time to be determined during the period of the award, or as requested by the project sponsor; and (d) related research dissemination venues (conferences, symposia, summits, workshops, etc.) and publications of interest to the broader government and industry communities, including the DoD, as well as publication and dissemination of periodic research progress reports.

Overall, we sought *to identify, track, and analyze new ways and means for how best to articulate, tailor, and streamline the process for acquiring different kinds of secure OA systems that incorporate Web-based and mobile devices running widgets, apps, and mashups*. We seek to do so in ways that focus on software cost drivers and highlight opportunities for cost reduction through alternative software components or system configurations. This investigation is therefore applicable to complex software elements used in many kinds of component-based OA software-intensive systems as commonly found in academic, business, or government enterprises. Last, this acquisition research supports and advances a public purpose by investigating challenges arising from the adoption and deployment of OA software systems for Web-based or mobile devices, which is central to the ongoing deployment of emerging mobile system capabilities in academic, business, or government enterprises, which is a broad audience for our research [ScA14b, ScA14c].

Scientific Background

The move to OA systems represents a transition from the acquisition of monolithic systems to the acquisition of reusable system components that can be integrated to realize different configurations of a software product line for a specific application domain [BeJ10, GuC10, JoB11, ReB12, ScA12b, WoS11]. These components are acquired within a software ecosystem that is evolving towards component provisioning within open repositories, where components from different producers are available for selection, evaluation, and system integration [GuW12, Iba13, MartL11, ReB12,

ScA12a, ScA13b]. However, current scientific understanding of software system costs focus attention to estimating the cost of development for new proprietary CSS systems, that do not anticipate use of OSS, nor the replacement/substitution of CSS with functionally similar OSS components [MadB11].

OSS represents an integrated web of people, processes, and organizations, including project teams operating as virtual organizations. The “purchase price” for most OSS is “no cost” meaning it can be downloaded and used without additional software license fees, subject to compliance with the OSS component's license. Consequently, there is a basic need to understand how to identify an optimal mix of OSS and CSS components within OA systems, as well as how they reduce or increase the system costs during design, integration, deployment, and evolution when OSS components may be substituted for CSS components. However, the relationship among OA, OSS, security requirements, and acquisition is complex and evolving, so consequently it is poorly understood [cf. Sca09, Sca10, ScA11, ScA12b, ScA13c]. Subsequently, in 2007-08, we began by examining how different OSS licenses can encumber software systems with OA, which give rise to new requirements for how best to acquire OA software-intensive systems the employ OSS software elements [ScA08] during system design, integration, deployment, and evolution [ScA13a, ScA13b].

Our recent acquisition research efforts demonstrate it is both possible and feasible to develop OA systems that incorporate best-of-breed software components, whether proprietary CSS or OSS, in ways that can reduce the initial and sustaining acquisition costs of such systems.

We strongly believe that our research results are applicable to enterprise information systems, which are widespread throughout DoD and the U.S. government, as well as to command and control (C2) systems (e.g., [ReB12, ScB12, ScA13b]) and other defense systems. The audiences for our presentations included in Chapters 2, 3, and 4 in this Final Report Volume I, and the compiled and integrated materials we are developing, as included in Volume II of this Final Report, demonstrate our commitment to communicate our research results to large, diverse audiences. Doing so however requires new guidance, and ideally automated tools, for explicitly modeling and analyzing the architecture of an OA system during its development and evolution, along with annotating the architecture with software component license rights and obligations. Our results thus demonstrate a major technological advance in the acquisition and development of OA systems, as a breakthrough in simplifying software license analysis throughout the contracting activities. Creating similar advances for streamlining the acquisition process while reducing the costs of secure OA systems is the next breakthrough that is needed.

Acquisition Research Problems and Our Approach:

The core of our proposed technical approach was to investigate a closely related set of research questions through systematic empirical observation of current software cost and IP licensing practices for different kinds of common application and infrastructure software components. We then sought to formalize and comparatively analyze these

observations and practices into computational schemes that supplement our existing framework for modeling and analyzing the licensing and security requirements of OA software systems. In short, we sought to extend our formal software IP modeling scheme to incorporate software component cost elements, in ways derived from the answers to our four research questions. These four research questions follow from our accomplishments described above and in detail elsewhere [ScA13a, ScA14a, ScA15a]. Each is described in turn.

Research question 1: ***What are the emerging principles, practices, and sharing agreements of secure OA systems to be acquired through Multi Party Engineering in Adaptive Agile Ecosystems of software component apps, widgets, and mashups for mobile devices available within online markets or app stores*****?** Researchers and practitioners have identified various kinds of software component/system design, integration, deployment, and evolution costs: one-time purchase of license rights, subscription-like licensing, usage-based licensing; source code licensing, technical data licensing, service licensing, licensing of data generated while the system runs; licensing of software elements needed at run time, deployment time, distribution time, build time, design time; training and support needed for a system to be usable; evolution costs to maintain the current system; evolution costs to reach future versions; evolution costs to branch out into related systems to enable reuse; and so forth. Which of these costs matters to whom, when, where, and why, and which should people in the acquisition workforce be expected to track and manage, especially when Web-based or mobile devices are to be incorporated?

Research question 2: ***What is the best way to formally model and analyze the business models incorporated into new Web-based and mobile software component IP licenses, and "shared agreements" among participating ecosystem end-user organizations that are interested in utilizing mobile software components*****?** Software system architectural decisions influence overall costs, but in what way? A brute-force approach could estimate overall system life cycle costs for each candidate architecture. Guidance for an architectural decision then requires calculating this estimate for each alternative that the decision may produce. The question of whether more direct connections may be made from specific classes of costs to specific kinds of decisions is an open one. How are people in the acquisition workforce to make such decisions and realize predictable costs of alternative system architectures, especially within the context of Web-based or mobile devices supporting command and control system application?

It is clear that overall system life cycle costs cannot in general be evaluated without information about the context in which the system is expected to be developed, built, distributed, deployed, used and evolved. The effect of architectural decisions on overall acquisition life cycle costs necessarily takes place and is strongly influenced by the context in which the costs are incurred, and an important role is played by such questions as how many instances of the system are expected, how much and what kind of usage is expected, over what time period, with what level of training and support, with what expected future evolution, preferring which suppliers and ecosystems, and so forth. It will be necessary to characterize the kinds of context that are needed, and to express contexts in a way that is manageable for the people who

must collect or estimate the information as well as for the architects, acquisition officers, and others that will use it in making decisions that will affect costs. We foresee that choosing among the contexts will be in some sense as influential and important as choosing among the myriad architectural alternatives. In any case, making sensible architectural choices will not be practical without appropriate characterizations of the contexts in which their effects will unfold.

Research question 3: ***What are the implications for acquisition management of mobile software components (widgets, apps, and mashups) and the required knowledge or skill they require for acquisition management personnel?*** In our previous work we have focused on non-monetary license obligations. While some obligations appear in more than one license, and a few appear to be very widely distributed across licenses, in general each license requires its own distinct list of obligations in exchange for the rights offered. An important result from our research has been an approach for placing license obligations (and rights as well) in a partial order, based on subsumption among the classes of actions that satisfy the obligations. Using this we can show that one license obligation subsumes another, or stated informally that satisfying the first obligation necessarily satisfies the second obligation as well. The subsumption relationship among obligations makes reasoning about licenses a manageable task. The monetary obligations of various kinds that proprietary licenses impose must be brought into this partial order.

While costs of the same class are ordered based on the numeric value involved, there are (as we note above) different classes of costs such as purchase costs, subscription costs, per-seat costs, support costs, and so forth. We see several avenues that appear promising, such as (to list a few of the more obvious ones) comparing an outright purchase cost with a subscription cost summed over the expected lifetime of the system, or subscription costs over different periods by converting them to the costs for a specific time period of interest. The many classes of costs raise the need for approaches for comparing or if possible unifying them. In addition, costs and other non-monetary license obligations must be considered together in a useful fashion.

Research question 4: ***What new information technologies or IT concepts for acquisition can further streamline BBP processes, activities, and initiatives targeted to secure OA systems with Web-based or mobile devices?*** It is essential to collect and calculate information about the effect of architectural decisions on overall costs, but just as essential to be able to combine and present it in a way that provides usable architectural guidance. Research questions in this phase include identifying relevant architectural decisions, marshalling the cost information relevant to each decision, and evaluating alternatives in a way that allows architects to make good choices. While a total of overall system costs, including monetary costs and non-monetary obligations that must be met, is a fundamental and important criterion, we foresee that more focused information will also be useful. For example, certain kinds of architectural decisions affect the evolution of the system's software ecosystem in ways that may not directly translate into costs but still have a powerful effect on whether the system will thrive in the future, such as steering the system away from closed interfaces and proprietary solutions toward open interfaces and solutions that can available from a variety of suppliers.

Overall, investigating and developing answers to these four questions is the focus of our proposed effort for the 2015 project period. In particular, we sought to develop, document, and deliver our answers through research publications that were presented at the *2015 Acquisition Research Symposium* and elsewhere, as described later. We similarly sought to articulate these answers in ways that can ultimately contribute to the practice and guidance provided to the acquisition workforce. Effort targeting such practice emerged primarily through interactions with the Assembled Capabilities Working Group organized by The MITRE Corporation to support the C3CB Office within the DASD(A).

Inter-project research coordination

We continue to believe we are and have been extremely well positioned to leverage our recent research work and results [AlS10, ScA08, ScA11, ScA12a, ScA12b, ScA13a, ScA13b, ScA13c, ScA14a, ScA14b, ScA15a] with the effort described in this Final Report. We have continued to build on our current research efforts in OSS [e.g., Sca10] and software requirements-architecture interactions [ScA08, Sca09], as well as our track record in prior acquisition research studies. Similarly, we have found recent related research supported by the DoD addressing related issues in OSS [HiW10] also influences our proposed effort. In addition, our effort builds from and contributes to research on software system acquisition within the DoD, focusing on software reuse [MaS12], SPLs [GuW10, BeJ10], open innovation and emerging software component markets [GuW12]. We thus believe our complementary research places us at an extraordinary advantage to conduct the proposed study that addresses a major strategic acquisition goal of the DoD and the military services [DoDOSA11].

Prospects for longer-term Acquisition-related research

Most large academic, business, or government enterprises are orienting their major system acquisition programs around the adoption of an OA systems strategy. This in turn increasingly encourages the adoption, development, use, and evolution of OSS components within an overall OA system. This is especially true for command and control systems and other business enterprise systems that must support Web-based or mobile devices. Thus, there is a significant need for sustained research that investigates the interplay and inter-relationships between (a) current/emerging guidelines for the acquisition of software-intensive systems, and (b) how secure, reusable software product lines [MaS12, WoS11] that employ an OA incorporating OSS/CSS component products (e.g., widgets, apps, and mashups) and their production processes [ScA13b], are essential to improving the buying power and cost-reduction effectiveness of software-intensive program acquisition efforts. Thus, there is a significant need for sustained research that investigates the interplay and inter-relationships between:

- (a) current/emerging guidelines for the acquisition of software-intensive systems within the DoD community (including contract management and software development issues), as well as emerging research issues within the software engineering community that can contribute to and benefit the DoD community in

the near-future [ScA16];

- (b) how secure, reusable software product lines [MaS12, WoS11] that employ an OA incorporating OSS/CSS component products (e.g., widgets, apps, and mashups) and their production processes [ScA13b], are essential to improving the BBP and cost-reduction effectiveness of software-intensive program acquisition efforts; and
- (c) how (a) and (b) contribute to advances and new insights for how best to realize the Better Buying Power initiatives [Ken15] addressing open architecture systems that may incorporate open source software components and closed source software components that are subject to different, possibly conflicting Intellectual Property (IP) licenses and cybersecurity requirements.

Research Results

Our research studies and results are included in the remaining chapters of this Final Report as five individual research publications. Each is briefly described in turn.

The *first* publication, appearing as Chapter 2 in this Final Report, focuses attention to the key problem addressed by our acquisition research effort throughout 2015. This paper, Achieving Better Buying Power through Acquisition of Open Architecture Software Systems for Web-Based and Mobile Devices, was presented at the *12th Annual Acquisition Research Symposium*, in Monterey, CA, in May 2015 [ScA15a]. As this paper summarizes our key acquisition research results, we choose to highlight and summarize the key issues/findings here, while the report in Chapter 2 provides the details.

- Acquisition program managers/staff (including in-house legal counsel) may not understand how software licenses affect OA system design, and how OA system design affects software licenses.
- Software license obligations and rights propagate through system development life cycle activities in ways not well understood by system developers, integrators, end-users, or acquisition managers.
- Different acquisition programs within the DoD and other government agencies may independently re-interpret software component licenses to mean different obligations are implied and different rights are received.
- Failures to understand software license obligation and rights propagation can reduce DoD buying power, increase software life cycle costs, and reduce competition.
- Software producers often provide idiosyncratic, heterogeneous licenses that generally conform to common business models and common license types.
- In the presence of heterogeneously licenses software elements, it is unclear what kinds of trade-offs can/should software system integrators or program acquisition staff make in order to maximize overall system development agility and evolutionary adaptation address.
- Software IP license and cybersecurity obligations and rights must be tracked,

accounted, and managed, but it is unclear if the acquisition workforce is prepared to do so.

- The DoD and other government agencies would financially and administratively benefit from engaging the development and deployment of an open source, automated software obligations and rights management systems (SORMS) solution that could be standardized, disseminated, and deployed for use within acquisition program offices.
- It is unclear and likely too early to identify how best to cultivate and sustain DoD online storefronts and software ecosystem.
- All of these findings can be both *anticipated and mitigated* through action and careful investment that best enable BBP 3.0 compatible solutions.

Last, in this publication, we identify and recommend three key opportunity area for future acquisition research and development that can help mitigate the issues just identified, as well as help to further realize BBP objectives and goals [Ken15], as follows. First, we need to research and develop **worked examples of well-formed OA system architectures** that are appropriate for C2 system capabilities, and that accommodate Web-based apps, widgets, and mobile devices. Such OA system architectures should specify representative and standardized component interfaces. The examples should also include carefully specified shared agreements that account for different IP licenses and diverse business models of software producers, system integrators, and multiple end-user organizations who must collectively act in ways that enable agile development and adaptive evolution of demonstrable C2 system capabilities.

Second, we need robust **open source models of application security processes and reusable cybersecurity requirements** that account for exigencies in heterogeneous app/widget software ecosystems, account for software evolution dynamics, formation and continuous improvement of automation-compatible shared agreements, and more. These models should account for description of current process practices, prescription of required verification and validation activities and outcome (deliverable documents or online artifacts), and proscription of what tools/techniques to use, by whom, when, where, and how.

Third, we need precise **domain specific languages for specifying**, **and automated analysis tools for continuously assessing and continuously improving, cybersecurity and IP license requirements for dynamically evolving Web/mobile C2 system-based capabilities**. The DSLs needed must be able to specify and operationalize the shared agreements between different DoD organizations, government agencies, and commercial enterprises involved in producing, integrating, or evolving component-based OA C2 system capabilities.

Overall, none of these three opportunity areas for acquisition research require fundamental scientific breakthroughs—instead, they require applied acquisition research effort that is focused on producing practical results. But as such research does call for new technology solutions that can support diverse acquisition programs,

identifying the appropriate funding vehicle or agency is a challenge, and one that may be beyond the current level of resources available for the small-scale research efforts currently funded by the Acquisition Research Program. Consequently, this publication presents results that primarily address our research questions 3 and 4 identified above.

The *second* publication, appearing as Chapter 3 in this Final Report, focuses attention to the identifying emerging research issues in the Defense OA ecosystem [ScA16a]. In contrast to the other acquisition research publications in this Final Report, this publication seeks to engage the software engineering R&D community (i.e., software producers and system integrators) that support industrial practices in the Defense Community, and by extension, the Defense Acquisition Community. Specifically, this publication identifies six software engineering research challenges whose solutions we believe will both contribute to OA system development practices, as well as to new practices within the broader acquisition and Defense Community concerned with OA and BBP objectives. Consequently, this publication presents results that primarily address our research questions 1 and 2, and thus has been submitted for publication at the *Workshop on Software Engineering and Industrial Practice*, International Conf. Software Engineering, May 2016.

The *third* publication, appearing as Chapter 4 in this Final Report, focuses attention to the Life Cycle Activities for Acquiring Software-Based Assembled Capabilities [ScA15b]. This publication was developed and electronically circulated during the Summer 2015 within the Assembled Capabilities Working Group, to support the DASD(A) office for C3CB. This publication was produced in response to ongoing discussion of technical challenges arising in the proposed acquisition and development of new software elements that include Web-based or mobile devices in support of C3CB system applications. The *fourth* publication, appearing as Chapter 5 in this Final Report, focuses attention to the *Starting Assumptions on Life Cycle Activities for Acquiring Software-Based Assembled Capabilities*, Assembled Capabilities Working Group, Working Notes, electronically disseminated 28 July2015. It is a companion publication to that just described, and was circulated for comment, review, discussion and revision within the ACWG. Both of these two publications describe technical details that arose as we sought to take the findings from [ScA13a, ScA14a, ScA15a], make them relevant for use to different program offices participating in the ACWG, and to identify and disambiguate misunderstandings that were detected, all under the aegis of the ACWG in support of the DASD(A) C3CB office. Together, these two publications present results that address our research questions 1, 3 and 4.

The *fifth* and final publication, appearing as Chapter 6 in this Final Report, focuses attention to issues, challenges, and opportunities in OSS system development for Aerospace and Defense applications [ScA16b]. This publication constitutes an invited tutorial presentation (invited in Fall 2015) to be presented at the *2016 Ground Systems Architecture Workshop,* hosted by The Aerospace Corporation, Los Angeles, CA, on 29 February 2016. The target audience for this presentation is contractors, program managers, and others within the Defense Community who are engaged in the production and/or integration of software elements into OA systems, including those

for C3CB applications. This publication presents results that address our research questions 1, 2, 3 and 4. This publication is therefore the more comprehensive in scope, as well as describing and integrating our diverse acquisition research efforts up to this time.

Overall, we are grateful for the support and funding we have received that enabled our acquisition research to continue, and as documented in this Final Report.

Acknowledgements

Preparation of this report and all work products therein benefitted from a research grant, #N00244-15-1-0010 from the Acquisition Research Program at the Naval Postgraduate School, Monterey, CA. None of the content of this Final Report has been reviewed, approved, or endorsed by the ARP, NPS, US Navy, Department of Defense or any other government agency. The work presented is solely the responsibility of the authors.

References

[AlS10] Alspaugh, T.A, Scacchi, W., and Asuncion, H. (2010). Software Licenses in Context: The Challenge of Heterogeneously Licensed Systems, *J. Assoc. Information Systems*, 11(11), 730-755, November 2010.

[AlS13] Alspaugh, T.A. And Scacchi, W. (2013). Ongoing Software Development without Classical Requirements, *Proc. 21st Intern. Conf. Requirements Engineering*, Rio de Janeiro, BZ, 165-174, July 2013.

[An12] Anderson, S. (2012). Software Licensing – Smart Spending in These Changing Times, *CHIPS: The Department of the Navy's Information Technology Magazine*, July, 28-31.

[BeJ10] Bergey, J., & Jones, L. (2010). Exploring acquisition strategies for adopting a software product line approach. *Proc. 7th Acquisition Research Symposium*. Vol. 1, 111-122, Naval Postgraduate School, Monterey, CA.

[CoR14] Cochran, J. and Reed, H. (2014). DoD Widget Working Group Report, 13 March 2014.

[DISA12] Defense Information Systems Agency (2012). *DOD Open Source and Community Source Software Development in Forge.mil*, SoftwareForge Document ID – doc26066doc26066, http://bit.ly/16abVh1, accessed 30 October 2012

[DISA12a] Defense Information Systems Agency (2012). *Strategic Plan: 2013-2018*, Version 1.0, http://www.disa.mil/News/PressResources/2012/~/media/Files/DISA/About/Strategic-Plan.pdf, accessed 30 October 2012.

[DoDOSA11] Department of Defense Open Systems Architecture (2011). *Contract Guidebook for Program Managers*, Vol. 0.1, December, https://acc.dau.mil/OSAGuidebook

[GGM14] George, A., Galdorisi, G., Morris, M., and O'Neil, M. (2014). DoD Application Store: Enabling C2 Agility, *Proc. 19th Intern. Command and Control Research and Technology Symposium*, Paper-104, Alexandria, VA, June 2014.

[GMH13] George, A., Morris, M., Galdorisi, G., Raney, C., Bowers, A., and Yetman, C. (2013) Mission Composable C3 in DIL Information Environments using Widgets and App Stores. *Proc. 18th Intern. Command and Control Research and Technology Symposium*, Paper-036, Alexandria, VA, June 2013.

[GMO14] George, A., Morris, M. and O'Neil, M. (2014). Pushing a Big Rock Up a Steep Hill: Lessons Learned from DoD Applications Storefront, *Proc. 11th Annual Acquisition Research Symposium,* Vol. 1, 306-317, Naval Postgraduate School, Monterey, CA.

[GuC10] Guertin, N. and Clements, P. (2010). Comparing Acquisition Strategies: Open Architecture versus Product Lines, Vol. 1, 78-90, *Proc. 7th Acquisition Research Symposium*, Naval Postgraduate School, Monterey, CA.

[GSS15] Guertin, N.H., Sweeney, R., and Schmidt, D.C. (2015). How the Navy Can Use Open Systems Architecture to Revolutionize Capability Acquisition: The Naval OSA Strategy Can Yield Multiple Benefits. *Proc. 12thAnnual Acquisition Research Symposium, Monterey, CA,* NPS-AM-15-004, May 2015.

[GuW12] Guertin, N. and Womble, B. (2012). Competition and the DoD Marketplace, *Proc. 9th Acquisition Research Symposium*. Vol. 1, 76-82, Naval Postgraduate School, Monterey, CA.

[HiW10] Hissam, S., Weinstock, C.B., and Bass, L. (2010). On Open and Collaborative Software Development in the DoD, Vol. 1, 219-235, *Proc. 7th Acquisition Research Symposium*, Naval Postgraduate School, Monterey, CA.

[Iba13] Ibanez, L. (2013). Ozone Widget Framework required to be open source under congressional law, 5 March 2013, http://opensource.com/education/13/2/ozone-widget-framework , accessed 8 June 2013. Also see, *Ozone Widget Framework*, https://www.owfgoss.org/

[JoB11] Jones, L. and Bergey, J. (2011). An Architecture-Centric Approach for Acquiring Software-Reliant Systems, *Proc. 8th Acquisition Research Symposium*, Vol. 1, 32-49, NavalPostgraduate School, Monterey, CA.

[Ken15] Kendall, F. (2015). *Implementation Directive for Better Buying Power 3.0*, 9 April 2015, http://www.acq.osd.mil/fo/docs/betterBuyingPower3.0%289Apr15%29.pdf . Also see Defense Acquisition University, *Better Buying Power*, http://bbp.dau.mil/

[Ke12] Kenyon, H. (2012). DoD, Intel Officials Bullish On Open Source Software; Government-wide Software Foundation In The Mix, *AOL Defense,* October 2012.

[MaS12] Mactal, R., Spruill, N. (2012). A Framework for Reuse in the DoN. *Proc. 9th Acquisition Research Symposium,* Vol.1, 149-164, Naval Postgraduate School, Monterey, CA.

[MadB11] Madachy, R, Boehm. B., Clark, B., Tan, T., and Rosa, W. (2011). US DoD Application Domain Empirical Software Cost Analysis, *2011 Intern. Symp. Empirical*

Software Engineering and Measurement, Banff, Canada, 392-395.

[MarL11] Martin, G. and Lippold, A. (2011). Forge.mil: A Case Study for Utilizing Open Source Software Inside of Government, *Open Source Systems*, Springer, 334-337.

[ReB12] Reed, H., Benito, P., Collens, J., and Stein, F. (2012). Supporting Agile C2 with an Agile and Adaptive IT Ecosystem, *17th. Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-044, Fairfax, VA, June 2012

[RNC14] Reed, H., Nankervis, J., Cochran, J., Parekh, R., and Stein, F. (2014). Agile, Adaptive IT Ecosystem: Results, Outlook, and Recommendations, *Proc. 19th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-011, Arlington, VA, June.

[Sca09] Scacchi, W., (2009). Understanding Requirements for Open Source Software, in K. Lyytinen, P. Loucopoulos, J. Mylopoulos, and W. Robinson (eds.), *Design Requirements Engineering: A TenYear Perspective*, LNBIP 14, Springer Verlag, 467-494, 2009.

[Sca10] Scacchi, W. (2010). The Future of Research in Free/Open Source Software Development, *Proc. ACM Workshop Future of Software Engineering Research* (FoSER), Santa Fe, NM, 315-319.

[ScA08] Scacchi, W. and Alspaugh, T., (2008). Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense, *Proc. 5th Acquisition Research Symposium*, NPS-AM-08-036, Naval Postgraduate School, Monterey, CA, May.

[ScA11] Scacchi, W. and Alspaugh, T., (2011). Advances in the Acquisition of Secure Systems Based on Open Architectures, *Proc. 8th Acquisition Research Symposium*, Vol. 1, Naval Postgraduate School, Monterey, CA.

[ScA12a] Scacchi, W. and Alspaugh, T., (2012a) Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *J. Systems and Software*, 85(7), 1479-1494, July 2012.

[ScA12b] Scacchi, W. and Alspaugh, T., (2012b). Addressing Challenges in the Acquisition of Secure Software Systems with Open Architectures, *Proc. 9th Acquisition Research Symposium*, Vol. 1, 165-184, Naval Postgraduate School, Monterey, CA.

[ScA13a] Scacchi, W. and Alspaugh, T., (2013a). Streamlining the Process of Acquiring Secure Open Architecture Software Systems, *Proc 10th Annual Acquisition Research Symposium,* Monterey, CA, 608-623, May 2013.

[ScA13b] Scacchi, W. and Alspaugh, T. (2013b). Processes in Securing Open Architecture Software Systems, *Proc. 2013 Intern. Conf. Software and System Processes*, 126-135, San Francisco, CA, May 2013.

[ScA13c] Scacchi, W. and Alspaugh, T. (2013c). Challenges in the Development and Evolution of Secure Open Architecture Command and Control Systems, *Proc. 18Th Intern. Command and Control Research and Technology Symposium*, Paper-098, Alexandria, VA, June 2013.

[ScA14a] Scacchi, W. and Alspaugh, T. (2014a). Achieving Better Buying Power through Cost-Sensitive Acquisition of Open Architecture Software Systems. *Proc 11th Annual Acquisition Research Symposium*, Monterey, CA, NPS-AM-14-C11P07R01-036, May 2014.

[ScA14b] Scacchi, W. and Alspaugh, T. (2014b). Cost-Sensitive Acquisition of Open Architecture Software Systems for Mobile Devices, Invited Presentation, MITRE-ATARC Workshop on Challenges in Legal and Acquisition, *Federal Mobile Computing Summit*, Washington, DC, 19 August 2014.

[ScA14c] Scacchi, W. and Alspaugh, T. (2014c). Reasoning about the Security of Open Architecture Software Systems for Mobile Devices, Invited Presentation, *Federal Mobile Computing Summit*, Washington, DC, 20 August 2014.

[ScA15a] Scacchi, W. and Alspaugh, T. (2015a). Achieving Better Buying Power through Acquisition of Open Architecture Software Systems for Web-Based and Mobile Devices, *Proc. 12th Annual Acquisition Research Symposium*, Monterey, CA, NPS-SYM-AM-15-088, May 2015.

[ScA15b] Scacchi, W. and Alspaugh, T. (2015b). *Notes on Life Cycle Activities for Acquiring Software-Based Assembled Capabilities*, Assembled Capabilities Working Group, Working Notes, electronically disseminated 30 June 2015.

[ScA15c] Scacchi, W. and Alspaugh, T. (2015c). *Starting Assumptions on Life Cycle Activities for Acquiring Software-Based Assembled Capabilities*, Assembled Capabilities Working Group, Working Notes, electronically disseminated 28 July2015.

[ScA16a] Scacchi, W. and Alspaugh, T. (2016a). Beyond Open Architecture: Issues, Challenges, and Opportunities in Open Source Software Development (OSSD) for Aerospace and Defense Applications. *2016 Ground Systems Architecture Workshop,* The Aerospace Corporation, Los Angeles, CA, 29 February 2016.

[ScA16b] Scacchi, W. and Alspaugh, T. (2016). Emerging Research Issues in the Defense Open Architecture Ecosystem, *Workshop on Software Engineering and Industrial Practice*, International Conf. Software Engineering, May 2016 (submitted for publication).

[ScB12] Scacchi, W., Brown, C. and Nies, K. (2012). Exploring the Potential of Virtual Worlds for Decentralized Command and Control, *Proc. 17th. Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper 096, Fairfax, VA, June 2012.

[Tak12] Takai, T.M. (2012). *Department of Defense Mobile Device Strategy*, Version 2.0, Office of the DoD Chief Information Officer, May 2012. http://www.defense.gov/news/dodmobilitystrategy.pdf .

[WoS11] Womble, B., Schmidt, W., Arendt, M., and Fain, T. (2011). Delivering Savings with Open Architecture and Product Lines, *Proc. 8th Acquisition Research Symposium*, Vol. 1, 8-13, Naval Postgraduate School, Monterey, CA.

Chapter 2:

Achieving Better Buying Power through Acquisition of Open Architecture Software Systems for Web and Mobile Devices

Achieving Better Buying Power through Acquisition of Open Architecture Software Systems for Web and Mobile Devices

Walt Scacchi and Thomas Alspaugh
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA

Abstract

Many people within large enterprises rely on up to four Web-based or mobile devices for their daily work routines—personal computer, tablet, personal and work-specific smartphones. Our research is directed at identifying, tracking, and analyzing software component costs and cost reduction opportunities within acquisition life cycle of open architecture (OA) systems for such Web-based and mobile devices. These systems are subject to different intellectual property license and cybersecurity requirements. Our research goal is to create a new approach to address challenges in the acquisition of software systems for Web-based or mobile devices used within academic, business, or government enterprises. Acquisition personnel in such enterprises will increasingly be called on to review and approve choices between functionally similar open source software (OSS) components, and commercially priced closed source software (CSS) components, to be used in the design, implementation, deployment, and evolution of secure OA systems. We seek to make this a simpler, more transparent, and more tractable process. Finally, this acquisition research supports and advances a public purpose by investigating acquisition challenges arising from the adoption and deployment of secure OA software systems for Web-based or mobile devices.

Overview

The Department of Defense, other government agencies, and most large-scale business enterprises continually seek new ways to improve the functional capabilities of their software-intensive systems with lower acquisition costs. The acquisition of open architecture (OA) systems that can adapt and evolve through replacement of functionally similar software components is an innovation that can lead to lower cost systems with more powerful functional capabilities. OA system acquisition, development and deployment are thus seen as an approach to realizing Better Buying Power (BPP) goals for lowering system costs, achieving technical excellence, enabling innovation, and advancing the acquisition workforce.

Our research identifies and analyzes how new software component technologies like apps and widgets for Web-based and/or mobile devices, along with their intellectual property (IP) license and cybersecurity requirements interact to drive down (or drive up) total system costs across the system acquisition life cycle. The availability of such new scientific knowledge and technological practices can give rise to more effective expenditures of public funds and improve the effectiveness of future software-intensive systems used in government and industry. Thus, a goal of this presentation is to explore new ways and means for achieving cost-sensitive acquisition of OA software systems, as well as identifying factors that can further decrease or increase the costs of such systems at this time.

We begin by briefly reviewing to identify a set of recent trends in the development of OA

software systems that intend to develop more capable OA systems. These trends include the transition to adoption of small-form factor software components as distinct applications (standalone and plug-in "apps") and widgets that exploit modern Web capabilities. We then turn to examine some key goals of the BBP 3.0 initiative [Ken14] that direct attention to adoption of OA system development practices that affect acquisition practices. Next, we identify a new set of emerging challenges to achieving BBP through OA software systems. We then identify three new practices to realize the cost-effective acquisition of OA Software systems.

Recent Trends Affecting Better Buying Power through OA Systems

We find there are four broad trends that mediate the cost-effectiveness and buying power of emerging OA system acquisition efforts. These include: (a) the move towards shared, multi-party acquisition and agile development of new OA systems across compatible software ecosystems; (b) exploitation of new software component technologies compatible with Web and mobile devices; (c) growing diversity of cybersecurity challenges to address during system development; (d) new software development business models for app/widget development and deployment. Each is examined in turn.

A. Multi-party acquisition and development system ecosystems – Many in the Defense community seek to embrace the acquisition and development of agile command and control (C2) and related enterprise systems [AGV14, GGM14, GMH13, GuW12, RBC12, ScA12b, ScA13c, Sca14a]. Such systems are envisioned to arise from the assembly and integration of system elements (application components, widgets, content servers, networking elements, etc.) within a software ecosystem of multiple producers, integrators, and consumers who may supply or share the results of their efforts. The assembly and integration of system elements produces "assembled capabilities for C2 systems" (AC-C2). Our purpose is to identify how our approach to the design of secure OA systems can be aligned with this emerging vision for agile C2 system development and adaptive deployment. We also focus on design of OA system capability involving office productivity and social media components [AGV14] that increasingly may be configured within a secure AC-C2 [ScA11, ScA12a, ScA13b].

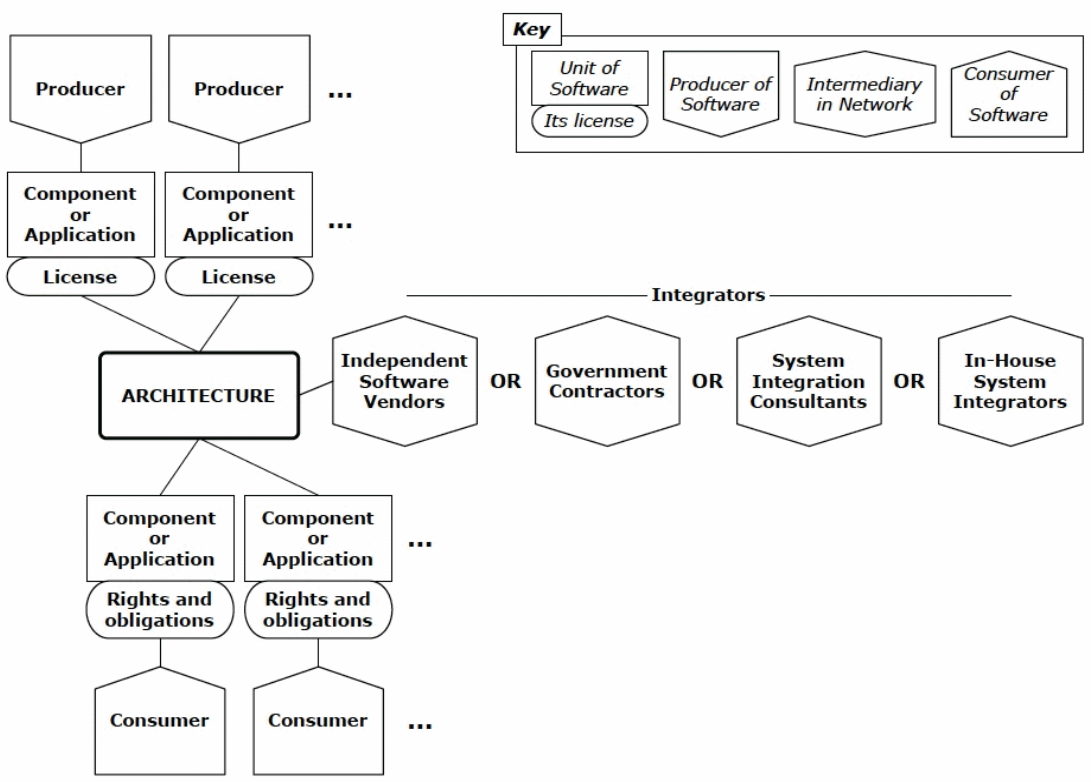
The design and development of agile C2 systems follows from two sets of principals: one set addressing guidelines/tenets for multi-party engineering (MPE) of C2 system components; the other set addressing attributes of agile and adaptive ecosystems (AAE) for producing AC-C2s or C2 system elements [RBC12, RNC14, ScA14a, ScA14b, ScA14c]. To help understand what we mean by a *software ecosystem*, we use Figure 1 to represent where different parties are located across a generic software supply networks or multi-party relationships that emerge to enable the software producers to develop and release products that are assembled and integrated by system integrators for delivery to end-user organizations, via online storefronts [GGM14, GMH13].

As noted, OA system components can include software applications (apps) and widgets. Widgets are lightweight, single-purpose web-enabled applications that users can configure to their specific needs [AGV14, Giz11, GMH13, ScA13b]. Widgets can provide summary information or a limited view into a larger application that can be used alongside related

widgets provides an integrated view, as required by users.

The lower part of Figure 1 also identifies where elements of shared agreements like IP licenses or cybersecurity requirements enter into the ecosystem, and how the assembly of components into a configured system or subsystem architecture by system integrators effectively (and perhaps unintentionally) determines which IP license or cybersecurity obligations and rights get propagated to consumer or end-user organizations. Agreement terms and conditions acceptable to consumer/end-user organizations flow back to the integrators. This helps reveal where and how shared agreements will mix, match, mashup, or encounter semantic mis-matches at the system architecture level, which is one reason why we use (and advocate) explicit OA system models.

Overall, a move towards MPE and AAE substantiates a path towards decentralized OA system development, integration, and deployment [DoD12, Giz11].



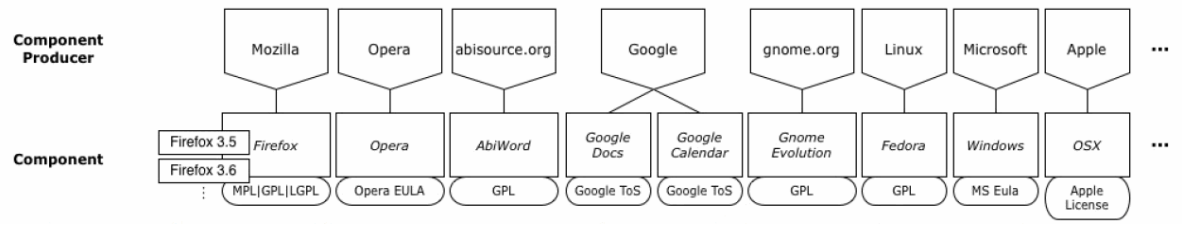


Figure 1. A generic software ecosystem supply network (upper part), along with a sample elaboration of producers, software component applications, and licenses for OA system components they employ (lower part) [ScA12a].

This decentralization will engender acquisition and development of *heterogeneously-licensed systems* (HLS), whereby different software components (apps, widgets) will be subject to different IP licenses [AIA13, AIS10], as well as to different cybersecurity requirements [DAG15, ScA12b, ScA13a, ScA13b, ScA13c]. This implies that such components, their IP licenses, and cybersecurity requirements will be subject to ongoing evolution across a diversity of methods, shown in Figure 2 [ScA12a, ScA13b]. These will create a new generation of challenges for the acquisition workforce, in terms of training, new work and contract management practices, and need for automated assistance to track and manage oversight of policy compliance (e.g., for alignment with BPP and cybersecurity assessment). Without automated assistance, it appears that the acquisition workforce will be overwhelmed with technical details that interact with acquisition, development, and/or system integration contracts and software component IP licenses and cybersecurity requirements. Otherwise, these conditions suggest that acquisition management practices can complicate acquisition [GGM14], and thus potentially mitigate the benefits of BBP that can arise from MPE and AAE for C2 systems.

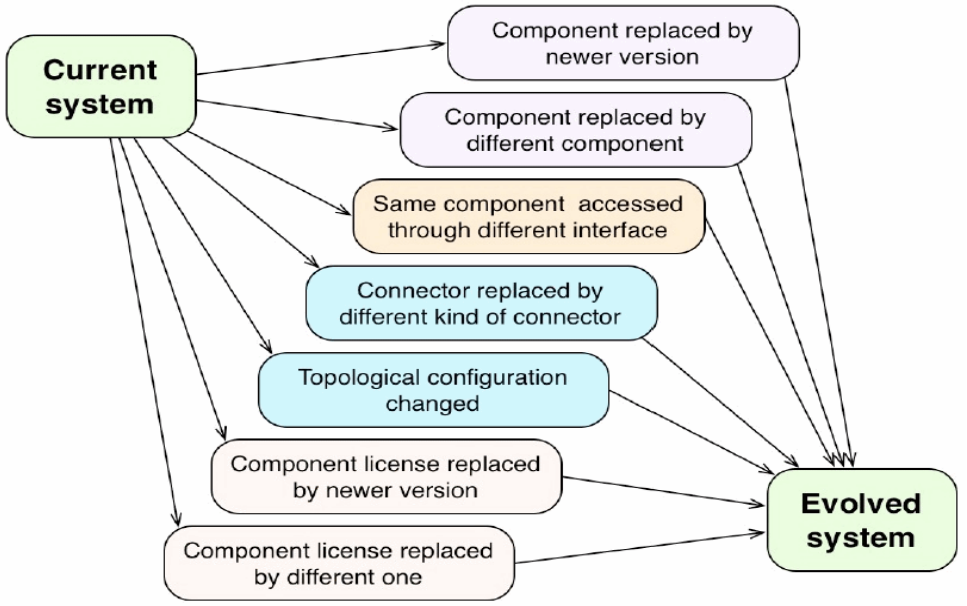


Figure 2. The kinds of common evolutionary changes that arise during OA software component development, deployment and sustained usage.

B. Moving towards shared development of Apps and Widgets as OA system components – Future OA systems for agile C2 may be configured by system integrators, end-user organizations, or war-fighters in the field. This would be accomplished through access to online repositories of software apps or user-interface widgets such as The Ozone Widget Framework (OWF), a government open source software (GOSS) effort that is central to such agile OA system development. The OZONE family of products includes the OWF and the OZONE Marketplace, the marketplace being an online repository whose operation is similar in kind to the online app stores by Apple and Google [ScA13b]. These products are built to fit the needs of human centered fusion activities in network centric warfare environments. The OZONE family of products is designed as a presentation layer toolkit

that can be rapidly deployed in a variety of mission contexts ranging from strategic planning to enable the creation of a real-time common operational picture and situation awareness applications. Figure 3 displays examples of OWF-based widgets operating in a Web browser, while Figure 4 shows OWF widgets deployed for use on a mobile device.

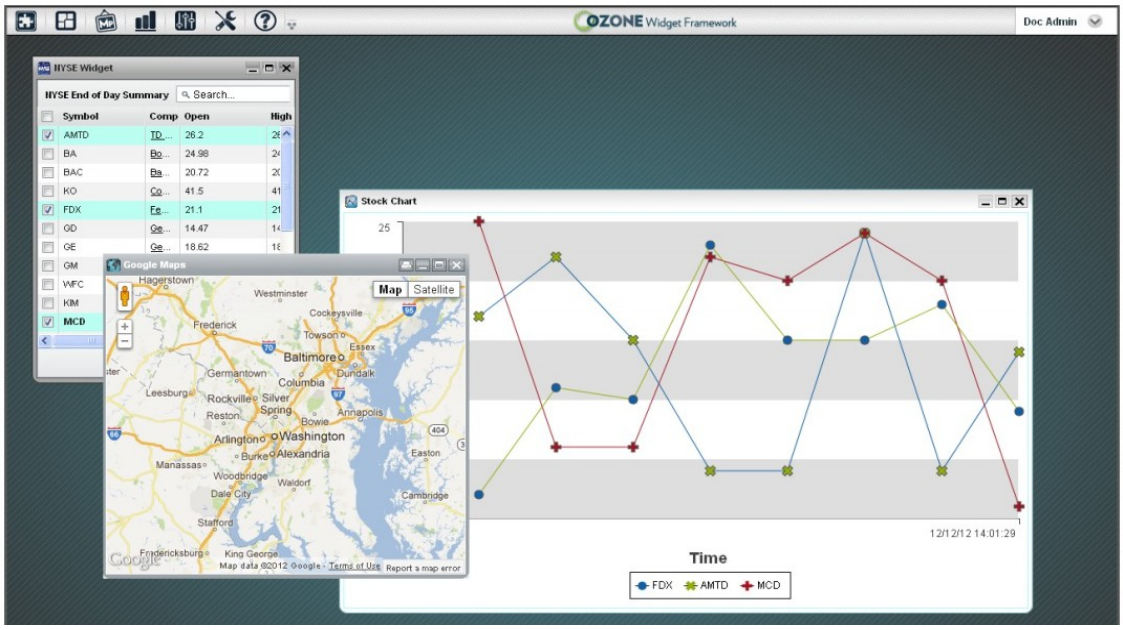


Figure 3. OWF Widgets running within a Web browser.

Figure 4. OWF Widgets running on a mobile device.

C. Growing diversity of challenges in cybersecurity – New types of software components like apps and widgets must be developed, deployed, and sustained in ways compatible with existing cybersecurity requirements. They must also be later adapted to accommodate emerging cybersecurity requirements that are not yet apparent. For example, there is growing interest in accommodating not just mobility, but also "Bring Your Own Device" (BYOD) capabilities.

BYOD suggests that end-users and war fighters are bringing their own mobile devices with themselves into the field to support their mission. However, BYOD clearly exacerbates the technical challenges of cybersecurity assurance, often in ways that cannot be readily anticipated, as when independently developed components co-evolve in conflict with one another [Wei14]. Nonetheless, acquisition policy necessitates cybersecurity vulnerability and exposures be addressed [DAG15]. But at present, it is unclear what new kinds of requirements these new OA system components bring to the acquisition workforce. For example, a move to adopt mobile apps and/or mobile widgets means these OA system components must pass though an application security process for "vetting" these components.

Vetting entails establishing what cybersecurity requirements are to be verified, how they are to be validated, as well as where, when and by whom these activities should be performed. One approach is to assume the vetting can be performed by a centralized authority, such as by the operator of the Ozone Marketplace. But it is not clear there will ever only be one such authority. Instead, if we foresee multiple marketplaces, which are already appearing both in GOSS and industrial online settings, then the acquisition workforce will be challenged in how best to determine which cybersecurity requirements must be addressed, validated, and compliance certified, as well as by whom and how often. Consider the example, seen in Figure 5, of a widget for "emergency response incident command system," developed for the Dept. of Homeland Security [Roc15]. How do its components (possibly GOSS) compare or interoperate with widgets/AC-C2 from DoD agencies or program offices concerned with C2 system interoperability or AC-C2?

Figure 5. AC-C2 style widget from the next-generation incident response system for DHS.

A move to widgets also presents new kinds of cybersecurity challenges when two or more widgets are configured together with one or more apps to create a mashup that provides an agile system capability. This situation refers to the technical challenges of inter-widget communication. Such component-component communication can be technically realized in different ways, such as via ad hoc, "open standards," or publish-subscribe messaging interfaces, as well as whether point-to-point or as configured through a dynamic processing mashup [CFG13, End13]. While OA system may rely on "open standards" style widget interfaces and communications patterns, widget communication/interface standards/ interfaces are still very new technologies and techniques. Thus, it is unclear which will survive and be widely adopted [End13a].

Similarly, knowledge about the proper usage of widget components is unclear, and thus is not yet ready for compliance assessment within current acquisition practices. The technical challenge is further complicated when apps/widgets are acquired from different online marketplaces. Different marketplaces may rely on different schemes for specification and interchange of shared data semantics between autonomously developed components. This in turn hinges on the expertise of OA system integrators, end-users, or war-fighters to recognize how, where, and when the semantics of technical data interchange arise and to what consequences via component-component API alignments (to avoid mis-matches), data type representations, data formats (e.g., "CSV" vs. .xls vs. XML), data naming conventions (for resource discovery vs. data modeling ontology), data range value limits, exceptional values, data-flow control signals, etc. These are still new technical problems that are yet to be readily resolved or to have development/usage guides.

D. New business models for OA software component development and use – New business models imply differentiated IP licenses and contracting practices. Given our discussion up to this point, along with reference to our recent acquisition research studies [AIA13, ScA11, ScA12b, ScA13b], this means different obligations and rights will be transferred from component producers to system integrators and end-user organizations. Some licenses are "buy and pay now," while others are "free now, pay later, based on usage," others are "many organizations (e.g., PEOs) will share purchase costs," and so forth.

Acquisitions of new kinds of OA system components allow for new business models. These include new models for software component producers, system integrators, and end-user organizations. For example, new software and OA system development business models for software app/widget development and deployment include (in no particular order): (1) franchising; (2) enterprise licensing; (3) metered usage; (4) advertising supported; (5) subscription; (6) free component, (7) paid service/support fees; (8) federation reciprocity for shared development; (9) collaborative buying; (10) donation; (11) sponsored development; (12) free/open source software (e.g., Government OSS – GOSS); and others [Hanf13]. Further, this list is not exhaustive; instead, it is only representative.

In contrast, for end-user organizations that involved in agile development of OA system components, or an integrated system capability, there is a need to develop and codify their own business models regarding OA software component development or system integration. These business models are constituted through "shared agreements" that allow for sharing

the cost of component or integrated capability development and cybersecurity assurance vetting across multiple parties (e.g., multiple Program Offices). However, these shared agreements are also a core part of emerging MPE/AAE development practices. These agreements must convey how OA component development or system integration costs and security assurance will be shared, as well as how they will be sustained in the presence of interacting software component development, deployment, and evolution processes and practices [ScA13a]. Shared agreements denote the obligations the participating organizations are willing to accept, in order to realize the provided rights they need. So shared agreements can be expressed and assessed in the same manner, and with the same analysis tools and techniques, as IP licenses and cybersecurity requirements [ScA13b, ScA13c].

Software acquisition costs easily become difficult to predict/manage given diversity of business models, IP licenses, and implied software component cybersecurity assessment. Development/usage cost sharing agreements can further complicate determination of development cost, costs shares across organizations, and system costs over time as business models, component licenses, and cybersecurity assessment requirements evolve [ScA12a, ScA13a].

What kind of expertise do we expect the acquisition workforce to need in order to make adoption of “component-based system capabilities” (including for mobile devices) agile, adaptive, and practical across different commercial/governmental software marketplaces/ecosystems? What kinds of acquisition guidance is needed for articulating and streamlining Shared Agreements between multiple organizations participating in shared OA component development and cybersecurity assurance? What kinds of acquisition management practices and analysis tools are needed for the acquisition workforce to insure cost savings and BBP in such settings? Addressing these questions is beyond the scope of this paper, but these questions require follow-on acquisition research to resolve and answer.

Better Buying Power 3.0 Goals

Better Buying Power (http://bbp.dau.mil/) is part of DoD's initiative that sees continuous improvement as the best approach to improving the performance of the defense acquisition enterprise. BBP 3.0 [Ken14] identifies eight areas of focus that group a larger set of itemized initiatives that offer the potential to restore affordability and realize technical excellence in defense procurement and improve defense industry productivity. One of the eight areas focuses on promoting or increasing competition, and this area includes an initiative to utilize modular open system architectures to stimulate innovation [Ken14]. Technical innovations are constrained by two categories of Intellectual Property (IP) rights available to the Government: (a) technical data (TD – e.g., product design data, computer databases, computer software documentation); and (b) computer software (CS – e.g., source code, executable code, design details, processes, and related materials). These rights are realized through IP licenses provided by system product or service providers (e.g., software producers) to the Government customer, so long as the customer fulfills the obligations stipulated in the license agreement (e.g., to indicate how many software users are authorized to use the licensed product or service according to a fee paid).

As already noted, our acquisition research has focused on issues addressing OA systems

and IP licenses since 2008 [ScA08], as well as forward to the acquisition of secure OA systems for command and control (C2) and enterprise information systems [ScA11, ScA12b, ScA13b], where security requirements can be expressed in a manner similar to IP obligations and rights. Therefore, here we turn to identify how a sample of different goals of BBP 3.0 initiatives interact or relate to the trends and challenges examined so far in this paper. The BBP goals are highlighted, and then followed by a brief examination.

- *Promote effective competition* – One central purpose for acquiring OA systems is to increase the likelihood of creating and maintaining competitive environments among system producers who can provide software components that can be replaced by similar offerings by other component producers. We demonstrate how this can work when system architectures are explicitly modeled, and their software components and interconnections are similarly specified in an open manner [AIA13, ScA12a]. Such openness also supports improved technology search and outreach, but enabling retrieval of compatible OA system components from online (software app) storefronts.

- *Use Modular Open Systems Architecture to stimulate innovation* – Open system architectures that can accommodate common components from alternative producers requires that the components utilize standardized interfaces, whether in the form of open Application Program Interfaces (APIs), standard data exchange protocols, and standard data representations, formats, and meta-data, as well as utilization of open source software (OSS) components [ScA08]. But also noted earlier, app and widget components at present have a plethora of standardized interfaces, and it is unclear which will survive, be sustained, be widely adopted (inside/outside of DoD), and be evolved [End13a].

- *Increase small business participation and opportunities* – one way to increase competition in the realm of OA systems is to identify where smaller scale software applications (apps) or widgets can be utilized, which might be produced by innovative small businesses or startup ventures which dominate much of the online markets for Web-based or mobile device apps/widgets. Small businesses may further be advantaged by their utilization of shared OSS infrastructure components, platforms, or remote services, since large commercial contractors may not see sufficient profit margins to develop proprietary alternatives. So OA systems that accommodate OSS components that can integrate custom apps/widgets into innovative system capabilities (AC-C2), may then realize new opportunities for DoD customers. Other small business opportunities may similarly arise for such ventures that focus on emerging cybersecurity assessment or tool development services.

- *Improve our leaders' ability to understand and mitigate technical risk* – In looking forward, there is potential interest in seeing the BPP initiative evolve to also address risk as an implicit cost driver. This might allow or innovative ways and means to reduce emerging risks through accelerated or "look ahead" system acquisition and development approaches that emphasize increased reliance on rapid prototyping.

- *Increase the use of prototyping and experimentation* – The rapid development of Web-

based or mobile app mashups might be performed by appropriately trained end-users or war-fighters [AGV14, End13]. A move towards OA systems for Web-based and mobile devices that rely on apps/widgets retrieved from online marketplaces--apps composed through interpretive software program "scripting" and mashup techniques--is a clear example of this [End13, GMH13 GuW12, ScA13a]. Thus, it is not surprising to find such emerging techniques being investigated and assessed for possible production of new C2 capabilities [GGM14, GMH13, ScA13b].

- *Achieve dominant capabilities through innovation and technical excellence* – an overall summary of the current BBP initiative is focusing attention of how to make acquisition more agile, more innovative, and to develop a new generation acquisition workforce that can enact acquisition processes that are technically excellent—thin and flexible when needed, yet robust and cost-effective, while also being amenable to continuous improvement. This is indeed a real challenge to fulfill, and beyond the scope of what current acquisition practices are likely to achieve without targeted investment in acquisition improvement research. To be clear, one just needs to consider emerging opportunities (and potential asymmetric cybersecurity threats) that arise through the desire to develop next-generation AC-C2 that are to be composed from apps/widgets that can operate on Web-based/mobile devices. What are the best processes or practices for acquiring, developing, and sustaining deployed systems that are to be built using these new software technologies (e.g., apps/widgets for mobile devices)? How should these processes and practices be adapted to accommodate personal devices (e.g., Apple iPhones/iPads, Android phones/tablets, Blackberry 10 phones) that individual war-fighters, joint force troops, or contracted service providers bring with them into the battlespace? How must acquisition processes be best adapted to accommodate and rely on software supply chains that arise around consumer-oriented app marketplaces as possible ways/means for *doing more* (e.g., rapidly prototyping warfighter composable C2 app/widget mashups [GMH13]) *without more* (e.g., war-fighters who bring their own mobile computing devices for use in C2 contexts) [AGV14, GGM14]? Once again, these are critical questions to address and resolve through new acquisition research and supporting technology development.

Emerging Challenges in Achieving BBP through OA Software Systems for Web-based and Mobile Devices

The business models and IP licenses for software components are tightly coupled: software component licenses codify component producer business models. Said more simply, licenses codify business models. So different software business models imply different software license obligations and rights, and different license types reflect different possible business models. Licenses are generally recognized as contracts regarding IP expressed through terms and conditions that specify obligations and rights stipulated by the component's producer to enable/constrain what can be done with the component by its integrator or end-users. Understanding and assuring software IP obligations and rights is routinely a task for acquisition offices, and thus a task to be competently performed by the acquisition workforce.

Obligations (like purchase costs/fees paid, or to insure access to open source software code modifications) denote conditions, events, or actions imposed by a software producer (the

licensor) that must be fulfilled by the software integrator/customer enterprise (the licensee) in order to realize the *rights* identified in the licenses (right to use; right to distribute copies; no right to distribute modified copies, etc.). Note that software system integrators play a role is shaping the obligations and rights imposed on customer enterprises based on choices they make in how software component-based systems are designed, built, and deployed. So where/who does system integration matters, as does whether customer enterprises that acquire systems have policies that determine which software licenses (or business models) they will accept.

Similarly, we note that "cybersecurity requirements" can also be expressed and analyzed in terms of obligations and rights [ScA11, ScA12b]. This suggests the problems and solutions to software component IP license management will be similar in kind or form to those for cybersecurity assurance. Below, we just focus attention to software IP obligations and rights, though the same consequences may apply to the cybersecurity of OA systems and components.

There are many unstated consequences that can arise when software licenses are not well understood. Here are some examples we have seen within the DoD context.

- *Acquisition program managers/staff (including in-house legal counsel) may not understand how software licenses affect OA system design, and vice-versa.* Component-based system design can determine which software licenses will fit, or which can fit if the system design is altered to encapsulate desirable software components with somewhat problematic license obligations or rights [ScA13a].

- *Software license obligations and rights propagate through system development life cycle activities in ways not well understood by system developers, integrators, end-users, or acquisition managers*. We have investigated and described many examples of this in a recent paper that shows how license constraints are mediated by software system design, build-integration, deployment, post-deployment support tools and activities.

- *Different acquisition programs within DoD and other government agencies may independently reinterpret software component licenses.* This realizes enterprise-wide inefficiencies, as well as increases avoidable costs. It appears to be technically possible to codify software component licenses by type or producer, especially with regards to performative obligations and operational rights that Program Offices or customer organizations seek. The license modeling techniques we have investigated demonstrates the potential, practicality, and scalability of such possibility [AIA13, ScA12a, ScA12b, ScA13b]. However, it may be most efficient and most effective for DoD to have common legal interpretations for different licenses (or different business models). Such interpretations could be common, if produced by a central legal authority (e.g., Office of General Counsel). Alternatively, it may also be possible for DoD and other government agencies to provide an open framework or (acquisition) policy guidance whose purpose is to encourage software producers to not only provide software licenses in current narrative forms, but also to provide them in computer processable forms (using domain-specific languages) amenable to automated license

analysis. Once again, this is a form of guidance and training we can provide, but it is not one that we can impose on anyone. We believe it is in the best interest of DoD and other government agencies to employ software licenses that are both human readable and formally processable though automated means, at least in terms of software license obligation and right determinations.

- *Failures to understand software license obligation and rights propagation can reduce DoD buying power, increase software life cycle costs, and reduce competition*. Guidance from the OUSD for Acquisition, Technology, and Logistics recommends programmatic adoption of different BBP 3.0 initiatives grouped into eight focus areas of relevance as methods for innovation, continuous improvements, and doing more without spending more. Acquiring licensed software components is a cost-generating activity, whose costs/fees can be reduced while acquiring evermore agile and adaptive software components and open architecture component-based systems. However, software license non-compliance or worse, infringement, on the part of DoD will generate costs, program delays, as well as reduce agility and adaptation, all of which can be avoided. Such situations can and must be avoided through acquisition and development practices with little/no additional cost to affect. Such practices can be codified within open source business processes or open source computational business process models that can be shared, customized to specific program needs, redistributed and archived [ScA13b].

- *Software producers often provide idiosyncratic licenses that generally conform to common business models and common license types.* This seems mainly to arise from efforts by software producers to protect or update their business models in ways that improve their financial yield or protect/lock-in their customer base. This in turn generates demand for time, attention, and effort from legal counsel that support acquisition programs, while also reducing the effectiveness and timeliness of program acquisition efforts. DoD and other government agencies may be able to explicitly specify in advance what kinds of generic software license obligations they will accept and what kinds of generic software rights they seek, through their own explicit business models. Such specifications can be codified and provided to software producers in open source manner through software license acquisition policies. Software producers might then separate license terms and conditions that do and do not address current license acquisition policies, in order to streamline licensing design and analysis practices for the mutual benefit of software producers, integrators, and customers.

- *Software producers generally provide software licenses that are assumed to legally dominate in systems composed of components from different software producers or integrators.* We refer to software systems (or systems of systems) composed from components (e.g., apps, widgets) subject to different licenses as “heterogeneously-licensed systems” (HLS) [AlS10, AlA13]. Popular Web browsers that are compatible with widgets, apps, or plug-in components (e.g., Google Chrome, Mozilla Firefox, Apple Safari) are subject to dozens of component licenses. Popular COTS software components also sometimes encompass components subject to multiple licenses. In both situations, the component producer asserts overall component license obligations

and rights in ways that are compatible with the licenses included therein (or so we hope). But when we deploy components that are composed into complex system architectures, or employ components that support on-demand download and implicit integration of smaller components (widgets, plug-ins, scripts, etc.) from online stores, then analysis of license obligation and rights propagation or encapsulation matters. Such technical details can readily overwhelm program acquisition managers and legal staff, thereby reducing the agility and adaptation of component-based system development/deployment. Provision of automated license analysis capabilities within software license management systems should be able to overcome this situation.

- *Given the challenges of HLS, it is unclear what kinds of trade-offs can/should software system integrators or program acquisition staff make in order to maximize overall system development agility and evolutionary adaptation address.* This situation is not unique to DoD, but is in fact widespread. However, as DoD and other government agencies move to embrace agile and adaptive component-based software systems to realize new, more timely system capabilities at lower cost compared to legacy approaches, then there is need to provide guidance for how to identify and manage such trade-offs. Failure to recognize the challenges of analyzing and managing HLS systems translates into opportunities lost while avoidable costs increase. We can and should do better than this. But this will require that resources be allocated to identify, articulate, train, and iteratively refine best practices about how, where, when, and why these trade-offs arise. Such knowledge should therefore be captured, codified, shared, accessed, updated, and redistributed in an open source manner.

- *Software IP license and cybersecurity obligations and rights must be tracked, accounted, and managed*. A move to component-based open architecture systems increases organizational overhead for managing software licenses. This overhead can be reduced, or better transformed into productive, value-adding business practices, through use of automated software obligations and rights management systems (SORMS). While SORMS exist and are routinely used by software component producers (to keep track of who has a licensed copy of their software products), SORMS do not exist at this time for software system integrators or customer enterprises.

- *DoD and other Government agencies would financially and administratively benefit from engaging the development and deployment of an open source automated SORMS.* This may represent the lowest cost means for simplifying license analysis while maximizing the benefits of agile and adaptive component-based software systems acquisition within the DoD and other government agencies. SORMS can help to better DoD software buying power. Similarly, an open source SORMS would also be of value to smaller or startup software producers who may best be able to create innovative and agile software components (widgets) in cost-competitive ways. Last, an open source SORMS intended for software integrator/customer enterprises would be of value to large, established DoD software producers, as a medium through which larger-scale software component acquisitions (e.g., components acquired for standardized deployment throughout an enterprise can be negotiated and simplified.

- *How best to cultivate and sustain DoD online storefronts and software ecosystem*. The acquisition of development of some DoD Web/mobile widgets may be strongly influenced by commercially available apps that are not secure, nor DoD information assured. War-fighters and others are often drawn to the best available technologies, including apps found in commercial online stores. Who decides whether apps in these conditions should be migrated, secured, and assured to meet DoD requirements? Alternatively, allowing such apps to be used as widgets for rapid prototyping new DoD AC-C2 may represent a promising new direction to stimulate innovation. Subsequently, this entails the needs to better understand possible commercial-DoD online storefront interactions and interdependencies, as well as articulating the needs of DoD Agency/Program Office-specific storefronts. Next, we expect to see redundant app offerings across multiple storefronts, including challenges of identifying common apps of different versions or variants across storefronts and user devices (e.g., is Google Maps the same version across all platforms in use; is Apple Maps equivalent to Google Maps; is Google Earth compatible with NASA World Wind?). How best to determine when redundancy is good/bad for such apps/widgets is unclear and under-explored at this time. Last, as noted, software component apps/widget licenses and business models across DoD Software App Ecosystem are very diverse with unclear/unknown interactions and interdependencies. Business models are codified in Web/mobile app IP licenses (e.g., conferring right to use or EULAs) and cybersecurity requirements. Again, much remains here to investigate and resolve to best enable BBP 3.0 initiatives realized with Web-based and mobile software.

Finally, as suggested along the way, *all* of these consequences can be both anticipated and mitigated through action and careful investment that best enable BBP 3.0 compatible solutions.

New Practices to Realize Cost-Effective Acquisition of OA Software Systems for Web-based and Mobile Devices

The trends and concerns identified above point to substantial challenges in identifying what can be done to both realize cost-effective BBP for Web-based and mobile device software apps, and to do so in ways that enable and empower the acquisition workforce in the years ahead. Technology, better buying practices, new business models, new cybersecurity requirements all point to the need for future research and development of new acquisition support technologies, work processes, and guidance practices. The goal is to make sure that acquisition time and effort does not become the main cost and the main risk factor going forward on the path to agile OA Web-based or mobile compatible C2 system development, deployment, and sustaining system evolution.

At this point, we see at least three key areas of opportunity for future acquisition research and development. First, we need to research and develop ***worked examples*** of well-formed OA system architectures that are appropriate for C2 system capabilities, and that accommodate Web-based apps, widgets and mobile devices. Such OA system architectures should specify representative and standardized component interfaces. The examples should also include carefully specified shared agreements that account for different IP licenses and diverse

business models of software producers, system integrators, and multiple end-user organizations who must collectively act in ways that enable agile development and adaptive evolution of demonstrable C2 system capabilities.

Second, we need robust, ***open source models*** of application security processes and reusable cybersecurity requirements that account for exigencies in heterogeneous app/widget software ecosystems, account for software evolution dynamics, formation and continuous improvement of automation-compatible shared agreements, and more. These models should account for description of current process practices, prescription of required verification and validation activities and outcome (deliverable documents or online artifacts), and proscription of what tools/techniques to use, by whom, when, where and how.

Third, we need precise ***domain specific languages*** (DSLs) for specifying, ***and automated analysis tools*** for continuously assessing and continuously improving, cybersecurity and IP license requirements for dynamically evolving Web/mobile C2 system-based capabilities. The DSLs needed must be able to specify and operationalize the shared agreements between different DoD organizations, government agencies and commercial enterprises involved in producing, integrating, or evolving component-based OA C2 system capabilities.

Overall, what we call for is similar in kind to what we have already produced and applied in other software development domains, using then current technologies [JeS05, ScA08]. What we now call for is a reinvention and repurposing of these concepts, but in contemporary forms scaled and secured in ways that best meet the needs of the DoD Program Offices, acquisition program managers, and others in the acquisition workforce to best support BBP 3.0 initiatives for Web-based and mobile device software components (widgets, apps, plug-ins).

Conclusions

The Department of Defense, other government agencies, and most large-scale business enterprises continually seek new ways to improve the functional capabilities of their software-intensive systems. The acquisition of OA systems that can adapt and evolve through replacement of functionally similar software component applications (apps) and widgets is an innovation that can lead to lower cost systems through more agile system development and adaptive system evolution. Our research identifies and analyzes how new software component apps and widgets, their IP license and cybersecurity requirements, and new software business models can interact to drive down (or drive up) total system costs across the system acquisition life cycle. The availability of such new scientific knowledge and technological practices can give rise to more effective expenditures of public funds and improve the effectiveness of future software-intensive systems used in government and industry.

Our study reported in this paper also identifies a new set of technical risks that can dilute the cost-effectiveness of Better Buying Power efforts. It similarly suggests that current acquisition practices aligned with BBP can also give rise to acquisition management activities that can dominate and overwhelm the costs of OA system development. This adverse condition can arise through app/widget vetting, new software business models, opaque and/or underspecified acquisition management processes, and the evolving interactions of new

software development and deployment techniques. Unless proactive investment in acquisition research and development can give rise to worked examples, open source models, and new acquisition management system technologies, the likelihood of acquisition management dominating agile development and adaptive deployment of component-based OA C2 system capabilities is increased.

Overall, this paper serves to help describe and detail how Web-based and mobile device software component technologies, IP licenses, security requirements, business models and adaptive system evolution interact. It also highlights what policies, practices, or technologies within DoD and other government agencies can simplify or exacerbate OA system cost arising at different points in the acquisition life cycle. Our common goal is to increase the ways, means, and beneficial consequences of the transition to the cost-effective acquisition of Web-based and mobile device OA software systems whose acquisition, development, deployment, and ongoing evolution are agile and adaptive.

Acknowledgements

The research described in this report was supported by grants N00244-14-1-0030 and N00244-15-1-0010 from the Acquisition Research Program at the Naval Postgraduate School, Monterey, CA. No endorsement, review, or approval implied. This paper reflects the views and opinions of the authors, and not necessarily the views or positions of any other persons, group, enterprise, or government agency.

References

[AGV14] Agre, J.R., Gordon, K.D., and Vasiliou, M.S. (2014). Practical Considerations for Use of Mobile Apps at the Tactical Edge, *Proc. 19th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-035, Fairfax, VA, June 2014.

[AlA13] Alspaugh, T.A, Asuncion, H. and Scacchi, W. (2012). The Challenge of Heterogeneously Licensed Systems in Open Architecture Software Ecosystems, S. Jansen, S. Brinkkemper, and M. Cusumano (Eds.), *Software Ecosystems: Analyzing and Managing Business Networks in the Software Industry,* Edward Elgar Publishing, 103-120, Northampton, MA.

[AlS10] Alspaugh, T.A, Scacchi, W., and Asuncion, H. (2010). Software Licenses in Context: The Challenge of Heterogeneously Licensed Systems, *Journal of the Association for Information Systems*, 11(11), 730-755, November 2010.

[CFG13] Chudnovsky, O., Fischer, C. Gaedke, M. and Pietschmann (2013). Inter-Widget Communication by Demonstration in User Interface Mashups. *Web Engineering*, Springer-Verlag, Lecture Notes in Computer Science, Vol. 7977, 502-505.

[DAG15] *Defense Acquisition Guidebook* (2014). CVE--Common Vulnerabilities and Exposures. Chapter 13.7.3.1.4, accessed April 2015. https://acc.dau.mil/CommunityBrowser.aspx?id=492079#13.7.3.1.4

[DoD12] Department of Defense (2012). *Joint Operational Access Concept*, Version 1.0, 17 January 2012, http://www.defense.gov/pubs/pdfs/JOAC_Jan%202012_Signed.pdf

[End13] Endres-Niggemeyer, B. (2013). The Mashup Ecosystem, in *Semantic Mashups: Intelligence Reuse of Web Resources*, Springer, 1-50.

[End13a] Endres-Niggemeyer, B. (2013). Mashups Live on Standards, in *Semantic Mashups: Intelligence Reuse of Web Resources*, Springer, 51-89.

[GGM14] George, A., Galdorisi, G., Morris, M. and O'Neil (2014). DoD Application Store: Enabling C2 Agility. *Proc. 19th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-104, Fairfax, VA, June 2014.

[GMH13] George, A., Morris, M., Galdorisi, G., Raney, C., Bowers, A., and Yetman, C. (2013) Mission Composable C3 in DIL Information Environments using Widgets and App Stores. *Proc. 18Th Intern. Command and Control Research and Technology Symposium*, Paper-036, Alexandria, VA, June 2013.

[GuW12] Guertin, N. and Womble, B. (2012). Competition and the DoD Marketplace, *Proc. 9th Acquisition Research Symposium*. Vol. 1, 76-82, Naval Postgraduate School, Monterey, CA.

[Giz11] Gizzi, N. (2011). Command and Control Rapid Prototyping Continuum (C2RPC) Transition: Bridging the Valley of Death, *Proceedings 8th Annual Acquisition Research Symposium,* Vol. 1, Naval Postgraduate School, Monterey.

[Han13] Hanf, D. (2013). *MPE/AAE Business Model Framework Overview*. Mitre Corporation, personal communication, July 2013.

[JeS05] Jensen, C. and Scacchi, W. (2005). Process Modeling Across the Web Information Infrastructure, *Software Process--Improvement and Practice,* 10(3), 255-272, July-September 2005.

[Ken14] Kendall, F. (2014). *Better Buying Power 3.0 Interim Release*, http://www.acq.osd.mil/dpap/sa/Policies/docs/BBP_3_0_InterimReleaseMaterials.pdf, 19 September 2014.

[RBC12] Reed, H., Benito, P., Collens, J., and Stein, F. (2012). Supporting Agile C2 with an Agile and Adaptive IT Ecosystem, *Proc. 17th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-044, Fairfax, VA, June 2012.

[RNC14] Reed, H., Nankervis, J., Cochran, J., Parekh, R., Stein. F. and others (2014). Agile and Adaptive Ecosystem, Results, Outlook and Recommendations. *Proc. 19th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-011, Fairfax, VA, June 2014.

[Roc14] Rockwell, D. (2015). DHS Transfer Emergency-Response Tech. *Federal*

Computer Week, 3 April 2015. http://fcw.com/articles/2015/04/03/dhs-nics.aspx

[ScA08] Scacchi, W. and Alspaugh, T., (2008). Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense, *Proc. 5th Acquisition Research Symposium*, NPS-AM-08-036, Naval Postgraduate School, Monterey, CA, May.

[ScA11] Scacchi, W. and Alspaugh, T., (2011). Advances in the Acquisition of Secure Systems Based on Open Architectures, *Proc. 8th Acquisition Research Symposium*, Vol. 1, Naval Postgraduate School, Monterey, CA.

[ScA12a] Scacchi, W. and Alspaugh, T., (2012a) Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *Journal of Systems and Software*, 85(7), 1479-1494, July 2012.

[ScA12b] Scacchi, W. and Alspaugh, T., (2012b). Addressing Challenges in the Acquisition of Secure Software Systems with Open Architectures, *Proc. 9th Acquisition Research Symposium*, Vol. 1, 165-184, Naval Postgraduate School, Monterey, CA.

[ScA13a] Scacchi, W. and Alspaugh, T. (2013a). Processes in Securing Open Architecture Software Systems, *Proc. 2013 Intern. Conf. Software and System Processes*, San Francisco, CA, May 2013.

[ScA13b] Scacchi, W. and Alspaugh, T.A. (2013b). Streamlining the Process of Acquiring Secure Open Architecture Software Systems, *Proc. 10th Annual Acquisition Research Symposium*, Monterey, CA, 608-623, May 2013.

[ScA13c] Scacchi, W. and Alspaugh, T.A. (2013c). Challenges in the Development and Evolution of Secure Open Architecture Command and Control Systems, *Proc. 18Th Intern. Command and Control Research and Technology Symposium*, Paper-098, Alexandria, VA, June 2013.

[ScA14a] Scacchi, W. and Alspaugh, T. (2014). Achieving Better Buying Power through Cost-Sensitive Acquisition of Open Architecture Software Systems. *Proc 11th Annual Acquisition Research Symposium*, Monterey, CA, NPS-AM-14-C11P07R01-036, May 2014.

[ScA14b] Scacchi, W. and Alspaugh, T. (2014). *Cost-Sensitive Acquisition of Open Architecture Software Systems for Mobile Devices*, Invited Presentation, MITRE-ATARC Workshop on Challenges in Legal and Acquisition, Federal Mobile Computing Summit, Washington, DC, 19 August 2014.

[ScA14c] Scacchi, W. and Alspaugh, T. (2014). *Reasoning about the Security of Open Architecture Software Systems for Mobile Devices*, Invited Presentation, Federal Mobile Computing Summit, Washington, DC, 20 August 2014.

[Wei14] Weir, M. (2014). BYOD Topic: How Complicated Can Calendars Be? *J. Cybersecurity and Information Systems*, 2(1). 18-19.

Chapter 3:

Emerging Research Issues in the Defense Open Architecture Ecosystem

Emerging Research Issues in the Defense Open Architecture Ecosystem

Walt Scacchi and Thomas Alspaugh
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA

ABSTRACT
The U.S. Defense Community denotes an ecosystem of industrial system/component producers, system integrators and customer organizations. This community now embraces the need to utilize open source software and proprietary software in the systems or system components it acquires, designs, develops and deploys for a variety of reasons. But the long-term transition to embrace OSS components has surfaced a number of software engineering issues that require research-led approaches and solutions. In this paper, we identify and describe six issues areas now found in the Defense OSS ecosystem that pertain to (a) unknown or unclear software architectural representations; (b) how to best deal with diverse, heterogeneous software IP licenses; (c) how to address cybersecurity requirements; (d) challenges arising in software integration and release pipelines; (e) how OSS evolution patterns transform software IP and cybersecurity requirements; and (f) the emergence of new business models for software distribution, cost accounting, and software distribution. We use the domain of command and control systems with Web and mobile devices as our focus to help illuminate these issues along the way. We close with suggestions for how to resolve them.

Keywords
Architecture, Licensing, Processes, Cybersecurity, Evolution.

INTRODUCTION

The U.S. Defense Community, including the military services and civilian-staffed agencies, is among world's largest acquirers of commodity and bespoke (custom) software systems. This community further extends its reach and influence on a global basis through national treaties and international alliances through enterprises like NATO. Bespoke software systems are primarily provided and developed through the Defense/Aerospace industry, though most non-Defense industry providers of software systems, application or services (i.e., the mainstream software products/service industry) also provide their wares to Defense system acquisition or procurement enterprises. Such acquisitions often entail software procurement or development contracts valued in the the millions to hundreds of millions of USD [16]. Certain kinds of software engineering (SE) research problems arise at this scale of endeavor and economic value, which are not visible or are insignificant in smaller scale SE R&D efforts.

In this paper, we focus attention to that slice of this world that focuses on the development and deployment of software-intensive command, control, communication, cyber and business systems, (hereafter, C3CB). We further limit our focus to the most common software elements found in C3CB systems, so as to allow open discussion and broad exposure to emerging SE research issues in this arena. Specifically, we draw attention to issues surrounding the development, integration, and deployment of multi-version and multi-variant software systems composed from different open source

software (OSS) and proprietary (CSS) software elements or remote services [22,23], eventually including recent efforts to support Web-compatible services and/or mobile devices in C3CB. This focus provides exposure to systems composed from apps acquired through various acquisition regimes, including apps downloaded from Defense Community App Stores. We do not focus on Defense research programs whose purpose is to produce innovative software concepts, methods, prototypes or demonstrations, through academic research, unless as noted.

DEFENSE COMMITMENT TO OPEN ARCHITECTURE AND OSS

Historical Background

Interest within the U.S. Department of Defense (DoD) and military services in open source software (OSS) first appeared in more than 10 years ago [cf. 7]. More recently, it has become clear that the U.S. Defense Community has committed to a strategy of acquiring software-intensive systems across the board that require or utilize an "open architecture" (OA) which may incorporate OSS technology or OSS development processes [11,17] that can help Defense customer organizations to achieve better buying power [15]. Why?

According to Riechers [21], the Air Force saw that with its software-intensive systems, there is increasing complexity of the software (code) itself, they may be "held hostage" to proprietary legacy components, they seek more timely delivery of new solutions, and that acquisitions and requirements take too much effort. So the Air Force is moving towards an OA development approach that embraces open standards, open data, open APIs, best-of-breed OSS, and OSS development practices.

According to Brig. Gen. Justice [12,13], the Army sought to move away from closed source software (CSS), expensive software upgrades, vendor lock-in, and broadly exploited security weaknesses. Subsequently, the Army seeks to adopt OSS because it may realize direct cost savings (compared to proprietary closed source software), gain access to source code to better develop domain and IT expertise, enable the transition to contemporary Web services/technologies, and to enable rapid injection of innovative concepts from diverse R&D/IT communities into systems for tactical command and control systems, future combat systems, enterprise (business) systems, and others [30].

Last, according to Guertin [10], the Navy sought to mitigate the spiraling costs of weapon systems through adoption of OA [17], as well as the adoption of open business models for the acquisition and spiral development of new systems. This may therefore necessitate better alignment of the system requirements and program acquisition communities, as well as to better alignment of industry and academic partners who engage in software-focused research and development activities with DoD support.

There are now a number of policy directives within the Defense Community that formally recognize that OSS system elements can be treated as commercial-off-the-shelf (COTS) components, and that bespoke software system development projects will utilize an OA, unless otherwise justified and approved [17]. Thus, developing contemporary C3CB that incorporate both OSS and CSS elements is "business as usual." However, many legacy Defense industry contractors are hesitant about how best to engineer such OA/OSS systems. For example, does an OA system imply/require that its software architecture be explicitly modeled, be accessible for sharing/reuse (e.g., as a Reference Model), as well as modeled in a form/notation that is amenable to architectural analysis and computational processing? So now some ten or so years later, we can begin to identify what kinds of SE research issues can be observed and investigated within the Defense Community associated with its transition to OA systems and OSS software elements, specifically for Web and Mobile devices within the realm of C3CB.

OA, Open APIs, and OSS

OA seem to simply suggest software system architectures incorporating OSS components and open

application program interfaces (APIs). But not all software system architectures incorporating OSS components and open APIs will produce OA, since OA depend on: (a) how/why OSS and open APIs are located within the system architecture, (b) how OSS and open APIs are implemented, embedded, or interconnected, (c) whether the copyright (Intellectual Property) licenses assigned to different OSS components encumber all/part of a software system's architecture into which they are integrated, and (d) many alternative architectural configurations and APIs that may or may not produce an OA [cf. 1,24]. Subsequently, this can lead to situations in which acquisition contracts stipulate a software-intensive system with an OA and OSS, but the resulting software system may or may not embody an OA. This can occur when the architectural design of a system constrains system requirements—that is, what requirements can be satisfied by a given system architecture, when requirements stipulate specific types or instances of OSS (e.g., Web browsers, content management servers) to be employed, or what architecture style [6] is implied by given system requirements.

Application domain of interest: C3CB with Web or Mobile Devices

C3CB are common information system applications that support modern military operations at a regional, national, or global level. These applications may be focused to address common military mission planning, mapping, resource status tracking and scheduling, and performance activities through application sub-systems. However, closely related C3CB systems applications are also in common use within civilian/public safety agencies, public infrastructure/utility operations, live television and sports event broadcasting, massively multi-player online game operations centers, and even in high-end motorsports racing like Formula 1. So study of SE issues arising in the Defense Community and its software development efforts can inform awareness of similar issues in other non-Defense software system domains.

Modern C3CB applications are increasingly expected/planned to be composed from best-available software components, whether OSS or CSS. Furthermore, as smartphones, tablets and laptop computers are being brought into the workplace, so too is interest within the Defense Community into supporting the acquisition and development of Web-compatible widgets and mobile apps provided through an emerging ecosystem of component producers and system integrators, for configuration into secure OA C3CB systems [9,19,20,26,29]. Common software elements for such systems include Web browsers open to extensions like custom Map widgets, and remote content servers, email and calendaring, word processing, local/networked file servers, and operating systems. The data processed by the software may have high-relevance to military missions/operations, or may just be the daily grind of data manipulated by “productivity” applications which most of us use routinely to perform/enact our work assignments. Security has been mostly addressable through system isolation or “air gaps” to the outside world. But this is no longer common practice, and cybersecurity concerns have risen to the top of functional and non-functional requirements for all such C3CB applications, and new OA systems are now required to be secure by design, by implementation, and through release and deployment, as well as subject to independent testing and certification. Secure OA designs can then entail different schemes for encapsulating different (sets of) components, use of virtualization schemes, shims and wrappers, while encrypting data transfers and storage, and configuring multi-level system access capabilities. But, we have found examples of where different OA system designs and configurations propagate security obligations, privacy protections, and access rights are mediated/nullified by different software component IP licenses or system updates, identified later.

OA ECOSYSTEMS

In our view, software ecosystems denote a network of software component producers, system integrators, and customer organizations. In the Defense community, producers and integrators are commonly industrial entities (defense contractors), while customer organization are military program offices. Figure 1 presents an abstract view of a software ecosystem that associates software components or apps with their producers, system architectures with system integrators, and delivered

component or integrated application systems with their customer. We also add annotations to indicate that each component or app has its own software IP license, and that integrated systems delivered to customers come with some composition of IP license obligations and rights propagated through the system's OA.

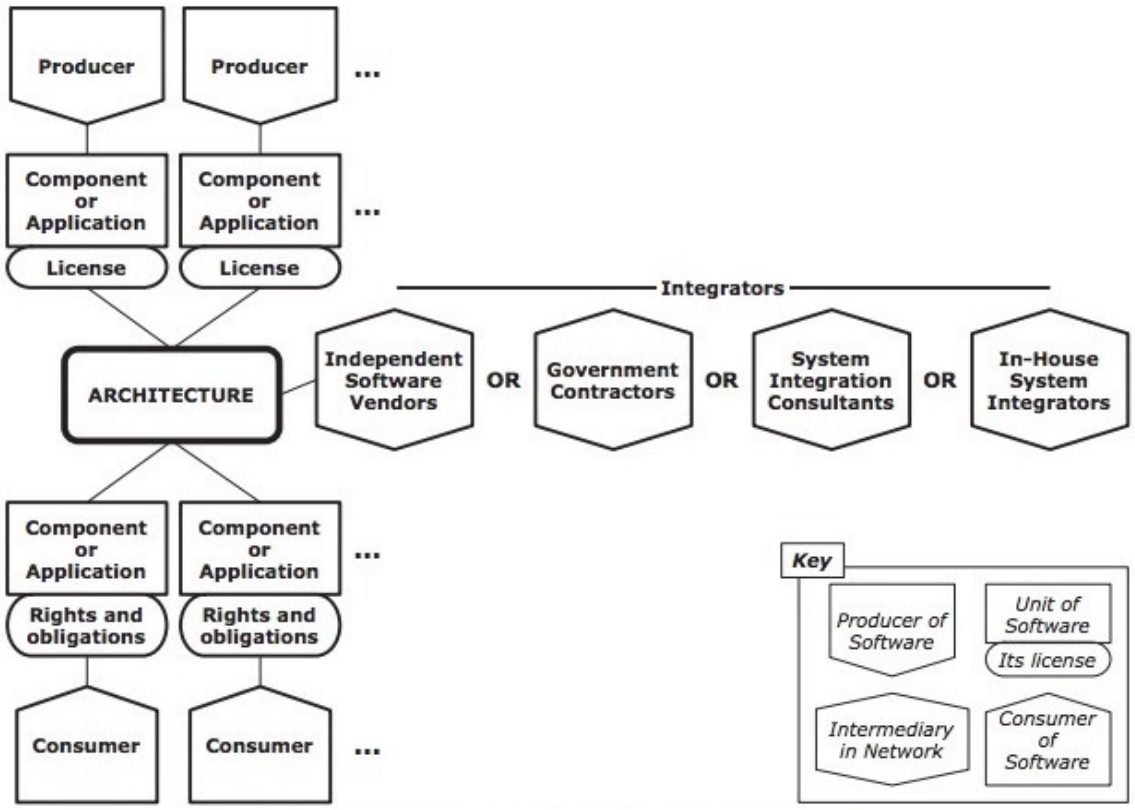


Figure 1. An abstract software ecosystem rendered as a software supply network (aka, a value chain).

There is growing interest within the Defense Community in transitioning to acquiring complex software systems via an agile and adaptive ecosystem [19,20,29], where components may be sourced from alternative producers or integrators, allowing for more competition, and ideally lower costs and better quality software elements that arise from a competitive marketplace [15]. But this adaptive agility to mix, match, reuse, mashup, swap, or reconfigure integrated systems or components requires that systems be compatible with, or designed to utilize, an OA—a software representation that identifies component types, component interconnections, and open APIs. An example of the common core of an C3CB system OA resembles most enterprise business systems, as C3CB are a kind of management information system for navigating, mapping, tracking resources; scheduling people and other resources; producing plans and documentation; supporting online email, voice or video communications. Figure 2 depicts an OA representation for such a kind of system. This OA representation can be read as a “reference model” for a C3CB software product line.

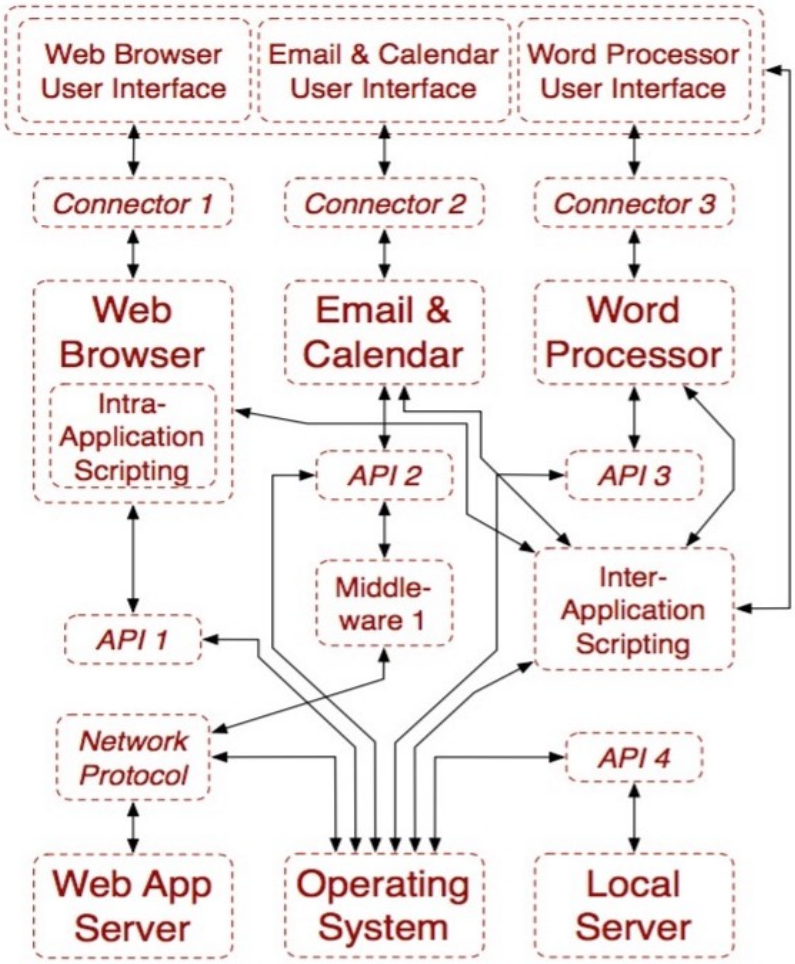


Figure 2. An OA reference model for common software component types interconnected within integrated C3CB systems.

One complication that can be anticipated here arises when component types are replaced with versioned component instances (e.g., Web Browser → *Firefox 40.0.3* or *Chrome 47.0.2526.111 (64-bit)*; etc.), where each component has a specific IP license (e.g., *Mozilla Public License 2.0* or *GPL 3.0*) associated with the versioned instance, which in turn may be viewed by system integrators as limiting an integrated system's architectural design, depending on how different components are interconnected in ways that may or may not propagate (un)desirable IP obligations and rights—a concern that arises frequently when using components subject to the GPL. As we have learned in practice, corporate lawyers employed by Defense contractors or in government do not have solutions for how to resolve such complexities, except via costly overall liability indemnification schemes, and efforts to distribute integrated systems with mostly IP obligations and few rights that effectively make an integrated open source system closed. This in turn can defeat the potential opportunities and benefits for commitment to OA systems that integrate OSS components.

With this background and sets of concepts for understanding a simplified view of the world of C3CB software systems, we now turn to identify and examine a set of issues that are now recurring in the acquisition, design, development, and deployment of such systems.

EMERGING SE RESEARCH ISSUES IN DEVELOPING AND DEPLOYING OA C3CB SYSTEMS WITH OSS ELEMENTS

We identify six kinds of emerging SE research challenges or issues that we have observed within the U.S. Defense Community as they have moved to OA systems for C3CB that utilize contemporary OSS and CSS components.

1. Unknown or unclear OA solutions

This first kind of challenge arises when acquiring new or retrofitting legacy software systems that lack an open or explicit architectural representation that identifies major components, interfaces, interconnections and remote services (if any). Though OA reference models are in use within the SE research community, contemporary C3CB generally lack such descriptions or representations that are open, sharable, or reusable. This may be the results of legacy business practices that see *software architectures as proprietary IP*, even when OSS components are included, or when applications sub-systems are entirely made of OSS code. An alternative explanation reveals that complex software systems like common Web browsers (Mozilla Firefox, Google Chrome, Apple Safari, Microsoft Internet Explorer) have complex architectures that integrate millions of SLOC that are not well understood, and that entail dozens of independently-developed software elements with complex APIs and IP licenses that shift across versions [25], such that the effort to produce an explicit OA reference model is itself a daunting architectural discovery, restructuring, and continuous software evolution task [8,14]. Thus, new ways and means for extracting software components interconnections and interfaces and transforming them into higher-level architectural representations are needed.

2. Heterogeneously licensed OA systems

OSS components are subject to widely varying copyright, end-user license agreements, digital civil rights, or other IP protections. The Open Source Institute recognizes dozens of OSS licenses are in use, through the top 10 represent more than 90% of the open source ecosystem [18]. This is especially true for OSS components or application systems that incorporate source code from multiple, independent OSS development projects, such as found in contemporary Web browsers like Firefox and Chrome which incorporate components from dozens of OSS projects, most with diverse licenses [25]. This means that OSS application systems are subject to complex software IP obligations and rights that may defy tracking, or entail legally contradictory obligations/rights [2]. Determining overall IP obligations for such systems is generally beyond the scope of expertise for software developers, as well as most corporate lawyers. Furthermore, we have observed many ways in which IP licenses interact within an OA software system, such that different architectural design choices that configure a fixed set of software components results in different overall system obligations and rights. Understanding multiple license interaction and IP mis-matches is way too confusing for most mortals and is a source of legal expense, or alternatively requires expensive indemnification insurance policies by the software producers or system integrators. Nonetheless, in our view, OA software ecosystems are defined, delimited, and populated with *niches* that locate specific integrated system solutions [25]. Furthermore, we see that these niches effectively have *virtual IP licenses* that must be calculated via the obligations and rights that propagated across integrated system component licenses via union, intersection, and subsumption relations among them [5]. Such calculation is daunting, and begs for a simpler, tractable, and computationally enforced scheme that can scale to large systems composed from many components. In such a scheme, OSS/CSS licenses could formalize IP obligations as operational requirements (i.e., computationally enforceable, at the integrated system level) instantiated by system integration architects. Similarly, customer/user rights are then non-functional requirements that can be realized and validated as access/update capabilities propagated across the integrated system [4].

3. Cybersecurity for OA systems

Cybersecurity is a high priority requirement in all C3CB systems, applications, and platforms [26,28]. No longer is cybersecurity something to be addressed after C3CB are developed and deployed—cybersecurity must be included throughout the design, development, deployment, and evolution of C3CB. However, the best ways and means for addressing cybersecurity requirements are unclear, and oftentimes somewhat at odds with one another depending on whether cybersecurity capability designs are specific to a: C3CB platform (e.g., operating system or processor virtualization; utilization of low-level operating system access control or capability mechanisms); component producer (secure programming practices and verification testing); system integrator (e.g., via use secure data communications protocols and data encryption); customer deployment setting (mobile: air-borne or ship-board; fixed: offices, briefing rooms, operations centers); end-user authentication mechanisms; or acquisition policy (e.g, reliance on third-party audit, certification, assurance of system cybersecurity). However, in reviewing these different arenas for cybersecurity, we have found that the cybersecurity requirements or capabilities can be expressed in much the same way as IP licenses: using concise, testable formal expressions of obligations and rights. Some examples follow (capital letters are placeholders that denote specified system, service, or component contexts).

- The obligation for a user to verify his/her authority to see compartment T, by password or other specified authentication process.
- The obligation for a specific component to have been vetted for the capability to read and update data in compartment T.
- The obligation for all components connected to specified component C to grant it the capability to read and update data in compartment T.
- The obligation to reconfigure a system in response to detected threats, when given the right to select and include different component versions, or executable component variants.
- The right to read and update data in compartment T using the licensed component.
- The right to replace specified component C with some other component.

These examples show how cybersecurity requirements can be expressed or paraphrased into/from restricted natural language (e.g., using a domain-specific language) into composite specifications that denote “security licenses” [2,3]. In this way, it should be possible to develop new software analysis tools whose purpose is to interpret cybersecurity obligations as operational constraints (executable) or provided capabilities (access control or update privileges), through mechanisms analogous to those used for analyzing software licenses [2,5], and how component or sub-system-specific obligations and rights can be propagated across a system's architecture. Consequently, we believe that cybersecurity can therefore in the future be addressed using explicit, computational OA representations that are attributed with both IP and cybersecurity obligations and rights.

4. Build, Release, and Deployment (BRD) Processes and Process Automation

C3CB applications represent complex software systems that are often challenging to produce, especially when conceived as bespoke systems. To no surprise, acquisition of these systems often requires a development life cycle approach, though some system elements may be fully-formed components that are operational as packaged software (e.g., commercial database management systems, Web browsers, Web servers, user interface development kits/frameworks). C3CB development is rarely clean-sheet and less likely in the future. Subsequently, component-based system development approaches are expected to dominate, thus relegating system integrators (or even end-users) to perform any residual source code development, inter-app integration scripting, or intra-app extension script development. But software process challenges arise along the way [27].

First, is again the issue noted earlier of whether there is an explicit, open source OA design

representation, preferably one that is not just a diagram, but instead is expressed in an architectural design language. With only a diagram or less, then is little/no guidance for how to determine whether a resulting software implementation is verifiable or complaint with its OA requirements or acquisition policies, such as provision or utilization of standardized, open APIs, intended to increase software reuse, selection of components from alternative producers, or post-deployment system extensions [15].

Second, is the issue arising from system development practices based on utilization of software components, integrated sub-systems, or turn-key application packages. These software elements come with their own, possibly unknown requirements that are nonetheless believed to exist and be knowable with additional effort [4]. They also come with either OSS code or CSS executables, along with their respective APIs. These components must be configured to align with the OA specification. Consequently, software tool chains or workflow automation pipelines are utilized to build and package internal/external executable, version-controlled software releases. We have found many diverse automated software process pipelines are used across and sometimes within software integration activities [27]. These pipelines take in OSS code files, dependent libraries, or repositories (e.g., GitHub), build executable version instances that are then subjected to automated testing regimes that include simple “smoke tests” and extensive regression testing. Successful builds that eventually turn into packaged releases that may or not be externally distributed and deployed as ready-to-install executables. While this all seems modest and tractable, when one sees the dozens of different OSS tools used in different combinations across different target platforms, then it becomes clear that what is simple is the small, is a complex SE activity when the scale of deployment increases.

Another complication that is now beginning to be recognized within and across BRD processes and process automation pipelines arises in determining when and how different BRD tool chain versions/configurations can mediate cybersecurity requirements in the target system being built. We have seen when software builds and deployed released are assumed to integrate to functionally equivalent CSS components, which are not included in releases, due to IP restrictions. We have also observed and reported how functionally equivalent variants as well as functionally similar versions may or may not be produced by BRD tool chains, either by choice or by unintentional consequence. This in our opinion gives rise for the need for explicit open source models of BRD process automation pipelines that can be analyzed, reused, and shared, as well as systematically tested to determine whether release versions/variants can be verified and/or validated to produce equivalent or similar releases that preserve prior cybersecurity obligations and usage rights.

5. Software Evolution Practices Transmitted Across the OA Ecosystem

Software evolution is among the most-studied of SE processes. While formerly labeled as “software maintenance,” a profitable activity mediated through maintenance contracts from software producers to customers, the world of OSS development projects and practices suggest a transition to a world of continuous software development—one that foreshadows the emergence of continuous SE processes, or software life cycles that just keep cycling until interest falters or spins off into other projects. OSS development projects rely on OSS tools that themselves are subject to ongoing development, improvement, and extension, as are the software platforms, libraries, code-sharing repositories, and end-user applications utilized by OSS developers to support their development work. Developers entering, progressing, or migrating within/across OSS projects further diversifies the continuous development of the most successful and widely used OSS components/apps. This dynamism in turn produces many ways for how OSS systems, or OA systems that incorporate OSS components evolve.

Figure 3 portrays different software evolution patterns, paths, and practices we have observed arising with new C3CB applications [25]. Here we see paths from a currently deployed, executable system release, to a new deployed release—something most of us now accept as routine as software updates are propagated across the Internet from producers, through integrators, to customers and end-users.

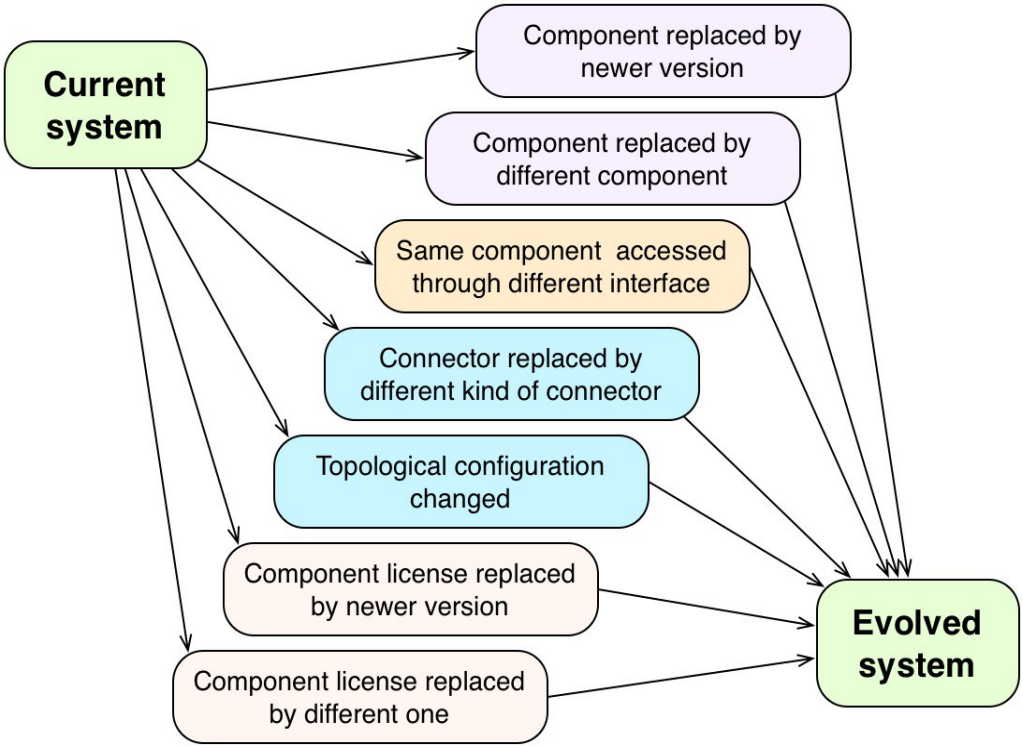


Figure 3. Different paths and mechanisms through which OA software systems can evolve [25].

Integrated OA systems can evolve through upgrades of functionally equivalent component variants (patches) as well as through substitution of functionally similar software components sourced from other producers or integrators. In Figure 4, we show a generic situation that entail identifying how an OA consistent with that depicted in Figure 2 may accommodate the substitution and replacement of a locally installed word processor application with a remote Web-based word processing software services (for example, Google Docs or Microsoft Office 365).

This is capability is a result of utilizing an OA that constitutes a reference model aligned with a vendor-neutral software product line. This is also a capability sought by customer organizations, and sometimes encouraged by software producers to accommodate their evolving business models (discussed below). While the OA remains constant, the location of the component has moved from local to remote/virtual, as has its evolutionary path. Similarly, the cybersecurity of the local versus remote component has changed in ways that are unclear, and entail a different, evolved assurance scheme.

Overall, the evolution of software components, component licenses, component interconnects and interconnections, and interconnected component configurations are now issues that call for SE research efforts to help make such patterns, paths, and practices more transparent, tractable, manageable, and scalable within an OA software ecosystem, as well as for customer organizations that seek the benefits of openness, sharing, and reuse.

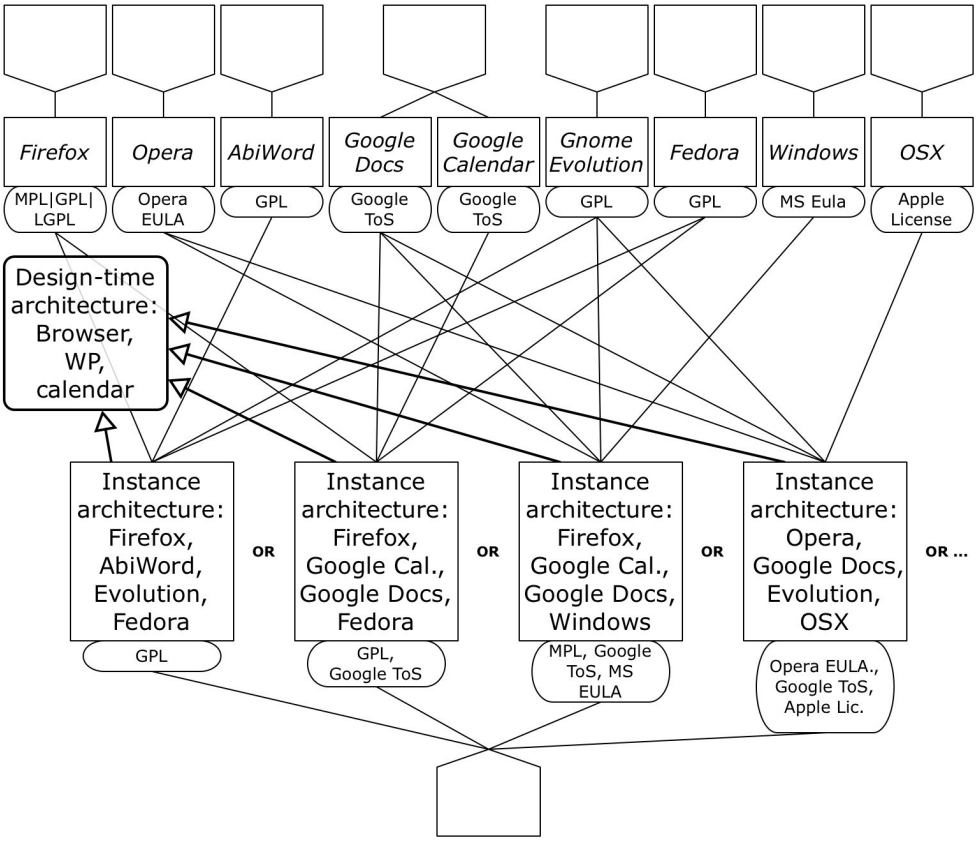


Figure 4. Alternative configurations of integrated instance releases of components consistent with the OA in Figure 2 that are treated as functionally equivalent by customer organizations.

6. New Business Models for Acquisition of Software Components and Apps

The last issue we address is the newest in this set of six for consideration for new SE research. While the field of SE research and practice has long paid attention to software economics, the challenges of software cost estimation are evolving in light of new business models being put into practice by software producers and system integrators.

In the past, software development projects were often managed by a single contractor responsible for both software production and system integration. Costs could be assessed through augmentation to internal business accounting practices (e.g., budgeting, staffing workloads, time-sheet reports, project schedules, etc.). But a move to OA ecosystems means that multiple producers can participate, and OA schemes accommodate switching among providers, while a system is being integrated, deployed, or evolved in the field. This in turn coincides with new ways and means to electronically distribute software updates, components, or applications, as well as new ways to charge for software. OSS components may be acquired and distributed at “no cost,” but their integration and evolution charged as service subscription, or as time-effort billings.

We have already seen other alternatives for costing or charging for software that include: franchising; enterprise licensing; metered usage; advertising supported; subscription; free component, paid service/support fees; federation reciprocity for shared development; collaborative buying; donation; sponsorship; free/open source software (e.g., Government OSS – GOSS); and others. So how are customer organizations, especially in the Defense Community where software cost estimation

practices are routine, suppose to estimate the development or sustaining costs of the software components or integrated systems they acquire and evolve, especially when an OA system allows for producers whose components come with different costing/billing schemes? This is an open problem for SE research in industry practice.

The last piece of the puzzle we are studying is the envisioned transition with the Defense Community to C3CB system being composed by customer organizations, and possibly extended by end-users deployed in the field. This is the concept that surrounds the transition to discovering software components, apps, or widgets in Defense Community app stores [9]. These app stores are modeled after those popularized for use in distributing and acquiring software apps for Web-based or mobile devices, like those operated by Apple, Google, Microsoft, and others. How the availability of such Defense Community app stores will transform the way C3CB systems are produced, or even if they will be produced by legacy Defense industry contractors remains to be seen. Said differently, how app stores transform OA software ecosystem networks, business models, or cybersecurity practices is an emerging challenge for SE research in industrial practice.

DISCUSSION and CONCLUSIONS

In this paper, we have focused attention to software engineering research challenges that are emerging in the industrial arena we called the Defense Community. Much of the earlier research and advances in SE emerged from challenges in this same community 40-50 years ago. Most contemporary SE research has moved away from this community. However, as we sought to describe in this paper, this community is again surfacing and facing a growing myriad of issues and challenges that can directly benefit from advanced in SE research.

We identified and examined six areas for SE research in industrial practice that now plague the Defense Community (and perhaps other industries as well). These six issues areas include: (1) the lack of architecture representations and schemes for discovering or specifying OA system designs; (2) OA systems that integrate components or applications subject to diverse, heterogeneous IP licenses; (3) how to manage the cybersecurity of OA systems during system design, development, and deployment; (4) software process challenges and evolving disruptions in seemingly mundane process automation pipelines; (5) software evolution patterns, path, and practices in OA ecosystems; and (6) how new business models are upending software cost estimation practices and outcomes. All of these SE research areas are readily approachable, and research results are likely to have significant practical value, both within the Defense Community and beyond.

These issue areas were investigated and addressed in the domain of command, control, communication, cyber and business systems (C3CB). We believe are all tractable, yet dense and sufficient for both deep sustained research study, as well as for applied research in search of near-term to mid-term practical results.

In related work [29], we have called for specific R&D investments into the development of open source, domain-specific languages for specifying open architecture representations (or architectural description languages) that are formalizable and computational, as well as supporting annotations for software license obligations and rights. While ADLs have been explored in the SE research community, the challenges of how software architectures mediate software component licenses and cyber security requirements is an open issue, with practical consequence. Similar, ADL annotations that assign costs or cost models in line with new software business models is an open problem area. We have also called for R&D investment in new SE tools or support environments who purpose is to provide automated analysis and support of OA systems IP and cybersecurity obligations and rights, as new requirements for industrial practice in large-scale software acquisition, design, development, deployment, and evolution. Such environments are the automated tools that could be used to model,

specify, and analyze dynamically configurable, component-based OA software systems expressed using the open source architectural representation schemes or ADLs noted here.

Hopefully, this paper serves to help throw light into these otherwise dark corners of SE research that can inform and add benefit to industrial software practices.

ACKNOWLEDGMENTS

The research described in this report was supported by grants #N00244-15-1-0010 from the Acquisition Research Program, at the Naval Postgraduate School, Monterey, CA. No endorsement, review, or approval implied.

REFERENCES

[1] Alspaugh, T.A and Antón, A.I., (2007). Scenario Support for Effective Requirements, *Information and Software Technology*, 50(3), 198-220.

[2] Alspaugh, T.A, Asuncion, H.A., Scacchi, W. (2010). Software Licenses in Context: The Challenge of Heterogeneously Licensed Systems, *J. Assoc. Info. Systems*, 11(11), 730-755.

[3] Alspaugh, T.A. And Scacchi, W. (2012). Security Licensing, *Proc. Fifth Intern. Workshop on Requirements Engineering and Law*, 25-28, September 2012.

[4] Alspaugh, T.A. And Scacchi, W. (2013). Ongoing Software Development Without Classical Requirements, (with T. Alspaugh), *Proc. 21st. IEEE Intern. Conf. Requirements Engineering*, Rio de Janeiro, Brazil, 165-174.

[5] Alspaugh, T.A., Scacchi, W. & Kawai. R. (2012). Software Licenses, Coverage, and Subsumption, *Proc. Fifth Intern. Workshop on Requirements Engineering and Law*, 17-24, September 2012.

[6] Bass, L., Clements, P., and Kazman, R., (2003). *Software Architecture in Practice*, 2nd Edition, Addison-Wesley Professional, New York.

[7] Bollinger, T., (2003). *Use of Free and Open-Source Software (FOSS) in the U.S. Department of Defense*, The MITRE Corporation, 2 January.

[8] Choi, S.C. & Scacchi, W. (1990). Extracting and Restructuring the Design of Large Systems, *IEEE Software*, 7(1), 66-71.

[9] George, A., Morris, M., Galdorisi, G., Raney, C., Bowers, A., & Yetman, C. (2013). Mission composable C3 in DIL information environments using widgets and app stores. In *Proc. 18th Intern. Command and Control Research and Technology Symposium*, Paper-036. Alexandria, VA.

[10] Guertin N., (2007). *Naval Open Architecture: Open Architecture and Open Source in DOD*, Presentation at "Open Source - Open Standards - Open Architecture," Association for Enterprise Integration Symposium, Arlington VA, 14 March 2007.

[11] Herz, J.C. And Scott, J., (2007). COTR Warriors: Open Technologies and the Business of War, *The DoD Software Tech News*, 10(2), 3-6, June.

[12] Justice, Brig. General Nick (2007a). *Open Source Software Challenge: Delivering Warfighter Value*, Presentation at "Open Source - Open Standards - Open Architecture," Association for Enterprise Integration Symposium, Arlington VA, 14 March.

[13] Justice, Brig. General Nick (2007b). *Deploying Open Technologies and Architectures within Military Systems*, Presentation at 3rd DoD Open Conference, Deployment of Open Technologies and Architectures within Military Systems, Association for Enterprise Integration Symposium, Arlington VA, 12 December.

[14] Kazman, R. and Carriere, S.J. (1998). Playing Detective: Reconstructing Software Architecture from Available Evidence. *J. of Automated Sofware Eng*., 6(2), 107-138.

[15] Kendall, F. (2014). Better Buying Power 3.0: Interim Release, 19 September 2014. Also see Defense Acquisition University, *Better Buying Power*, http://bbp.dau.mil/

[16] Meyers, B.C. and Obendorf, P., (2001). *Managing Software Acquisition: Open Systems and COTS Products*, Addison-Wesley, New York.

[17] OA (2016). Open Systems Architecture, https://acc.dau.mil/CommunityBrowser.aspx?id=1801

[18] OSSI (2016). The Open Source Initiative, http://www.opensource.org/
[19] Reed, H., Benito, P., Collens, J., & Stein, F. (2012). Supporting Agile C2 with an agile and adaptive IT ecosystem In *Proc.17th International Command and Control Research and Technology Symposium* (ICCRTS), Paper-044. Fairfax, VA.
[20] Reed, H., Nankervis, J., Cochran, J., Parekh, R., Stein. F., *et al*. (2014). Agile and adaptive ecosystem: results, outlook and recommendations. In *Proc. 19th International Command and Control Research and Technology Symposium* (ICCRTS), Paper-011. Fairfax, VA.
[21] Riechers, C., (2007). The Role of Open Technology in Improving USAF Software Acquisition, Presentation at "Open Source - Open Standards - Open Architecture," *Association for Enterprise Integration Symposium*, Arlington VA, 14 March.
[22] Scacchi, W. (2002). Understanding the Requirements for Developing Open Source Software Systems, *IEE Proceedings--Software*, 149(1), 24-39. Also see Scacchi, W. (2009). Understanding Requirements for Open Source Software, in K, Lyytinen, P. Loucopoulos, J. Mylopoulos, and W. Robinson (eds.), *Design Requirements Engineering: A Ten-Year Perspective*, LNBIP 14, Springer-Verlag, 467-494.
[23] Scacchi, W. (2010). The Future of Research in Free/Open Source Software Development, in *Proc. ACM Workshop on the Future of Software Engineering Research* (FoSER), Santa Fe, NM, 315-319.
[24] Scacchi, W. and Alspaugh, T.A. (2008). Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense, *Proc. 5th Acquisition Research Symposium*, NPS-AM-08-036, Naval Postgraduate School, Monterey, CA, May.
[25] Scacchi, W. and Alspaugh, T.A. (2012). Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *J. Systems and Software*, 85(7), 1479-1494, July 2012.
[26] Scacchi, W. and Alspaugh, T.A. (2013a). Challenges in the Development and Evolution of Secure Open Architecture Command and Control Systems, *Proc. 18th Intern. Command and Control Research and Technology Symposium,* Paper 098, Alexandria, VA, June.
[27] Scacchi, W. and Alspaugh, T.A. (2013b). Processes in Securing Open Architecture Software Systems, *Proc. 2013 Intern. Conf. Software and System Processes*, 126-135, May 2013, San Francisco, CA.
[28] Scacchi, W. and Alspaugh, T.A. (2013c). Advances in the Acquisition of Secure Systems Based on Open Architectures, in *J. Cybersecurity & Information Systems*, 1(2), 2-16.
[29] Scacchi, W. and Alspaugh, T.A. (2015). Achieving Better Buying Power Through Acquisition of Open Architecture Software Systems for Web-Based and Mobile Devices, *Proc. 12th Annual Acquisition Research Symposium*, Monterey, CA, May 2015.
[30] Starrett, E., (2007). Software Acquisition in the Army, *Crosstalk: The Journal of Defense Software Engineering*, 4-8, May, http://stsc.hill.af.mil/crosstalk.

Chapter 4:

Notes on Life Cycle Activities for Acquiring Software-Based Assembled Capabilities

Notes on Life Cycle Activities for Acquiring Software-Based Assembled Capabilities

Walt Scacchi and Thomas Alspaugh
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
wscacchi@gmail.com

Version of: 30 June 2015

1.0 Introduction

In this note, we identify and briefly describe the acquisition life cycle of assembled capabilities (AC) for software-intensive command, control, communications, cyber, and business systems (C3CB). Such systems may incorporate widgets, plug-ins, apps, and other mission components that accommodate Web-based or mobile devices.

Our overall goal is to work towards a new approach to acquisition of AC that is timely, produces technologically superior solutions, and does so in a cost-effective, agile, and adaptive manner. Said differently, our goal is how to avoid making well-intended AC acquisition practices become a strategic vulnerability. Consequently, our goal is to identify how best to provide new ways and means for streamlining AC acquisition through timely, agile, and adaptive technologically superior ways and means.

The remainder of this notes is organized into two sections: our views on the life cycle issues, risks and opportunities; and discussion and conclusions. There are also References included that provide further details that underlie or substantiate the materials presented. Each of these sections is presented in turn next.

2.0 Life Cycle Issues, Risks, and Opportunities

We now turn to identify issues, risks, and opportunities arising during the development, deployment and field adaptation life cycle of AC. The life cycle activities in focus include: procurement via shared agreements; requirements; design; integration and testing; release and deployment; installation and local configuration, and post-deployment adaptation and evolution. Each is addressed in this order.

2.1 AC Procurement –

There are new adoption challenges that can arise in the procurement of AC. These challenges arise due to emerging shared agreements among two or more parties (e.g., Program Offices) acting to share acquisition responsibilities or costs. Other challenges may arise due to different scenarios regarding technological constraints or opportunities afforded by integration if new innovative mission components with legacy systems or common technologies. As more than a dozen such challenges or use cases have already been identified by the ACWG/BMTT, then each would benefit from a systematic analysis of how these challenges/use cases are translated within the development, deployment, and support life cycle of relevant exemplar AC.

2.2 AC Requirements –

Traditional ways and means for articulating and specifying the requirements for C3CB capabilities are a key factor driving acquisition processes. How system requirements change or evolve is also a key factor (or the root cause) leading to the very long timelines associated with acquiring complex software-intensive system capabilities. Developing complex software is a difficult business and technological endeavor whose history is punctuated with far too many failure stories entailing cost and schedule overruns, compared with too few cost-effective, timely success stories. Why this is so is beyond the scope of discussion here, but suffice to say that if our goal is to realize agile, timely, and cost-effective acquisition of AC, that following current requirements engineering practices will not likely accomplish much beyond the status quo. Consider the following conditions that apply to acquiring AC.

2.2.1 Component/AC Mission Requirements: An AC has multiple types of requirements that are nominally suppose to be certified for completion or compliance by acquisition personnel. There are: (a) mission functional requirements (FR) that express functional operations or performance levels of an AC; (b) non-functional mission requirements (NFR) refer to necessary contextual elements that may or not part of the delivered AC, but are nonetheless required for the intended operation of the functional AC elements. Common NFR include specification of the types and/or versions of the C3CB platform that hosts the delivered AC, as well as cybersecurity and IP license fulfillment obligations (e.g., right to use software components as installed; right to modify and redistribute software components within an AC; only accept software updates from pre-specified trusted authorities). NFR specify what is needed to make a system or AC usable. In general, FR and NFR are often understated or implied by customers, yet are critical to successful use of deployed components or AC. While satisfaction of FR may be determined via operational validation of verified system functions through automated testing assistance, NFR traditionally assume human observation, intervention, or experience to demonstrate their fulfillment or satisfaction. However, FR and NFR are interdependent and thus jointly affect system design, integration, and deployment processes and decisions. Consequently, failure to identify and address NFR thus increases risk of operational system component or AC failure, or uncontrolled cost during development and deployment. Specifying sufficiently complete FR for software components or AC takes too much skill, effort, and time, thus increases the risk of difficult to control acquisition costs and unreliable delivered system performance. The currently best available way to avoid or mitigate such risks is to limit the scope of new development effort to those that utilize reusable components and interconnection mechanisms or AC that coalesce into tailorable product lines [GSS15, MaS12, ScA12, WoS11]. Such components or AC imply the need for reusable FRs, NFRs, and OA that can be tailored, but with knowable and measurable trade-offs. Such reuse can dramatically streamline acquisition, perhaps to the level of months.

2.2.2 Requirements versus Provisionments: Traditionally, specification of FRs for complex software-intensive systems was seen a critical to the system's robust, resilient, and reliable development and engineering. For example, many software systems have on average about 1000 source lines of code per FR, so that a system with 1000 requirements would likely necessitate a software implementation of 1,000,000 SLOC, and by extension 10K FR corresponding to 10M SLOC. But oftentimes, enterprise customers or Program offices have far fewer explicit FR, which producers necessarily transform into many more FRs. This is because those FR articulated by the customer are often underspecified, ambiguous, and stated in natural language that allows imprecision and hedging (e.g., “I will know what I need once I see it”, or “We need this because they already have it”). Imagine needing to specify the

FR for a modern Web browser, word processor, or slide presentation application that we now take for granted as being widely available and common to many C3CB environments. Such components are most likely simply to be specified by name or app type (i.e., specified as NFR), rather than in terms of the FR they fulfill. Such components have FR that are knowable in theory (e.g., via reverse engineering, user command menus or documentation), but are unknown in practice. Thus, we find ourselves in the position where you find yourself asking if anyone has explicit, documented FR for a Web browser or word processor that many C3CB utilize. This is a situation we cannot address or resolve.

In contrast, most/all AC for C3CB will incorporate one or more components that utilize maps of one kind or another. Mapping apps vary by type and integration of data that is fused from sources and sensors (e.g., fusion of ISR data) for overlay display on a map. Subsequently, map display and interaction utilities, along with repositories that serve user-requested data for display on the map, can be realized as a standalone app, plug-in extension for a Web browser, or as a widget., depending on the scope and scale of their functional capabilities and integration with legacy components, common repository technology and infrastructural services. So map-based modules are common mission components that can be tailored for use across multiple AC, if their common and mission-specific FR and NFR can be systematically elicited and parametrically coded.

Last, in the age of Web-based online stores (e.g., Apple App Store, GEOINT App Store, Google Play [ScA13b], and SPAWAR [GGM14, GMO14]), many software components are now simply based on what has already been delivered in some form. Such apps already appear within the fielded systems, commercial or consumer markets, or based on some innovative mashup that composes functional capabilities from multiple components or common technology services. This suggests that we are heading towards a world where software components accessed from online stores or repositories come with knowable but unknown FRs. This in turn implies we now prefer to simply determine whether the components available satisfy enough of our needs, or curiosity, to justify their adoption and usage. Consequently, we refer to the knowable by unknown FR as “provisionments,” meaning software components now provide functional capabilities to potential customers or end-users, and it is up to the customer or user to decide whether what is provided is satisfactory or close enough to meet unstated FRs in whole or part [AIS13].

The primary technical risks for provisionment-driven components usage in an AC include unknown cybersecurity vulnerabilities or reliability issues. Such risks may be mitigated through (a) prior widespread use in unsecured operation to reveal faults compared to the value realized of near-term availability for usage; (b) cybersecurity containers (e.g., virtual machines or operating system containers) that isolate the ability of a component to access, input/output data into/from mandatory access controlled system areas or storage mechanisms [ScA13d]; or (c) cybercurrency mechanisms that maintain a decentralized, encrypted but verifiable proof-of-work development update history that reveal where and when any vulnerabilities or unknown updates have entered into system executable code storage areas. Of these risk mitigations, (a) is most common but informal, so coverage depends on crowdsourced component usage; (b) is increasingly found in large enterprises actively engaged in cybersecurity defense; while (c) is a state of the art R&D challenge. Finally, it should be noted that the existence of such risks and common methods for their mitigation (i.e., (a) and (b)), suggests these conditions do not justify their avoidance and return to status

quo requirements practices: as noted, current practices may not really produce any better results, and that current practices are already known to be unreliable, costly and untimely, while producing sometimes unusable or limited performance systems. Nonetheless, provision-driven software components and their use within AC is a path towards improvement in acquisition.

2.3 AC Architectural Design and Mission Component Design –

There are two aspects of design process that are of interest: architectural (system-level) design and detailed (component-level) design.

2.3.1 AC architectural design – Assuming the requirement for use of an OA means there must be an explicit representation and method for specifying the architectural design of an AC. Such an architectural design configures a set of component types that are interconnected through explicit, open interfaces. Components include both those that are specific to the mission, and common technology and infrastructure services that are encapsulated as components accessed through public APIs. Components at the architectural design may be functional modules like an app found in an online store, or may be composition/configuration of multiple components/services that are grouped into a logical sub-system that may be shared or reused across different AC. This means that sub-systems can be treated from a design standpoint as just another component. So a system like the Secure Web Integration Framework from SPAWAR [DBA15] could be treated as either a composed system or a mission component for some AC. This flexibility in representation brings benefits of simplification through use of system abstractions such as OA and AC reference models that are also used to specify software product lines. For example, a “content management system” typically is composed of a multi-tier, Web-based repository that includes a Web server (like Apache) that serves content requested by user query, a database management system, (like MySQL or Oracle 11) that stores, queries and updates metadata associated with the requested content, and an underlying network file server (e.g., NFS, ZFS) that stores and updates the content (data, text, or media assets) requested by user query. Consequently, a CMS may be treated as a standalone system, or as a component (sub-system) in some AC. Components also may critically specify their interfaces through which data and control signals enter/exit, as well as any corresponding constraints. In particular, constraints of interest include IP licenses and cybersecurity requirements.

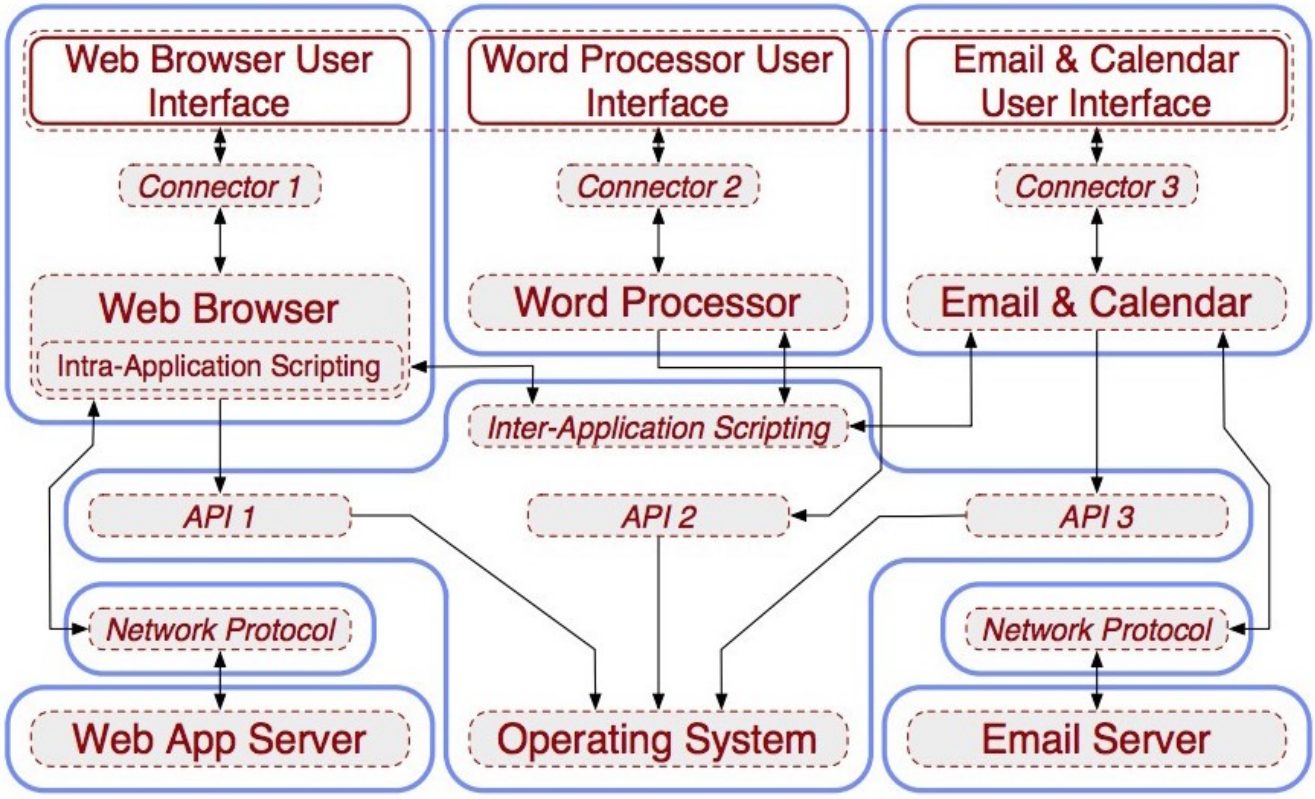


Figure 1. An OA design specification—a reference model—for a family of common networked C3CB applications utilizing typed components for Web browsing, word processing, email and calendaring, common technology and infrastructure services, each respectively within a security container boundary.

IP licenses may stipulate that components connected via APIs to other components APIs, can transfer IP rights and obligations across such interfaces. This condition may or may not be desirable, and thus requires guidance from the customer as to what IP obligations and rights they are willing to accept, or seek to propagate. Cybersecurity requirements work in a similar manner, so that customers often stipulate that connection of an un-secure component to another that has been assured to be secure, will require that such interconnection must not expose or provide access rights to the un-secure component, while the secure component may access and pull specified data (subject to further security checks) from the un-secure component. Again, the customer must be able to specify the cybersecurity requirements or enforceable policies (e.g., access control obligations and access rights). Thus, we need a method and explicit representational scheme to specify system or AC architectural configuration of components and interconnections. Ideally, such a scheme is also coupled to the AC implementation (e.g., software source code, connectors, and binary executables) such that evolutionary changes in configuration of the AC architectural design, integration and release package build, deployment and installation, and end-user run-time environment are visible, predictable, and tractable (perhaps even automatically propagated).

OA benefit strongly from their specification in an explicit architectural description language

(ADL). ADLs are used to identify and model the components within a composed system, system of systems, or AC, as well as their interconnections (what components exchanges data with what other components), in what is sometimes called the architectural layout (analogous to floor plans for a new building). In this regard, ADLs can have both textual and visual diagram views. Many ADLs exist, but choices for supporting, interactive architectural design environment, as an automated system for creating and updating ADL specifications for an AC are few. So there is a technology gap here that in turn reduces the potential benefits of Better Buying Power initiatives that seek to improve competition, improve innovation and increase adaptability of fielded systems.

Some technical risks can be identified that pertain to the architectural design of AC. These include:

- Closed architecture systems/AC with therefore unknown IP license and cybersecurity obligations and rights passed on to the customer/end-user.
- Closed architecture systems/AC may have vulnerabilities that are unknowable and difficult to reconstruct and overcome.
- Closed architecture systems/AC require the customer to trust and be locked into the system integrator, which works against improving competition and reducing costs.

As noted earlier, closed architecture software systems mitigate the benefits of OA and thus also mitigate against the desired outcomes from the Better Buying Power initiatives [Ken15]. Other technical risks that can arise during the architectural design of AC include:

- OA diagrams that do not call out explicit connectors, connection mechanisms, and component interface ports, which in turn are:
  - Vulnerable architectural mis-matches that prevent seemingly obvious system composition or AC that might otherwise be very desirable;
  - Vulnerable to neglect of man-in-the-middle or connector-based cybersecurity attacks.
- OA specifications that lack IP license and cybersecurity requirements annotations on components (interfaces) or overall system/AC may be unable to systematically determine the scope of obligations required and rights provided in response to evolutionary system changes.

2.3.2 Mission Component Design for Composition/Assembly within the OA for an AC

In simplest terms, the main element of component design to enable its integration and configuration within an AC is an explicit, open interface (open APIs). As noted, there is still legal debate now in progress before the SCOTUS as to whether APIs are (a) subject to license (so that their use requires a license agreement), (b) subject to fair use doctrine (thus open for use without license), or (c) not subject to copyright protection (which applies in Europe, and thus to NATO countries and AC for NATO systems). This is a mess, and represents a different kind of risk that we cannot address. However, back to the world we seek, it is desirable for mission components to provide explicit open interfaces which allows mission components to be integrated using open interconnection mechanisms. This is mostly an issue for components to be developed from scratch, and for components for common

technologies and infrastructure services, whether new or legacy.

2.4 AC Integration and Testing –

Continuous integration (CI) systems support automated processes for building, testing, and packaging a software system for release deployment. Without a CI system, developers must build, test, and integrate their software (component) products using hand-crafted scripts, and it is common for such scripts to have to rely on idiosyncratic dependencies on tool chains and libraries versions for each deployment platform targeted. In contrast, CI systems incorporate the capabilities of software build systems that may invoke sequential, distributed, or parallel builds across multiple build servers to produce singular builds (e.g., “nightly builds”), continuously updated agile development builds, or diverse, functionally equivalent executable variants. The build systems access and update software code (version control) repositories via process automation scripts. CI sub-processes take as input directories/folders of source code files and produce software component executables. The executables may also be organized as a structured collection (an information architecture) of binary files, static data value and parameter setting files packaged in interlinked directories, constituting releases for deployment.

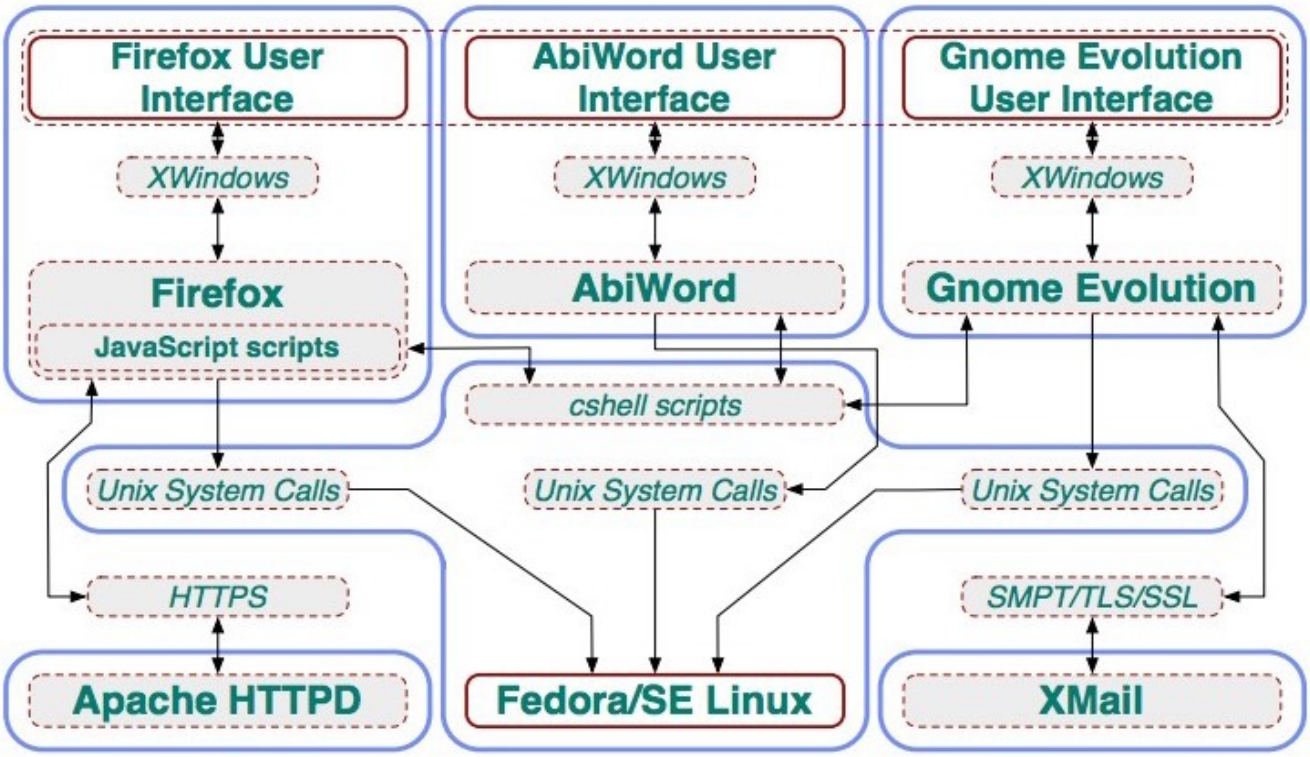


Figure 2. An Integration and Testing Build-time view of a specific integration of common components that are compatible with the OA design and cybersecurity containment scheme in Figure 1.

Automated CI systems comprise composed environments of software tools, or sets of loosely coupled tools together by automated process invocation scripts that guide and constrain their

use. Often these tools are independently developed and evolved. Automated CI environments are continuously being improved or supplanted, and different CI systems offer different features, functional capabilities, and depend on different software tools. So no single CI environment or approach will be universal, nor easy to standardize for use in AC multiple acquisitions, unless such standardization is stipulated within shared agreements, and release deployment is constrained to homogeneous platforms and component versions.

Next, one of the first activities in moving from architectural design to continuous integration is to identify specific software component versions that can be instantiated within the current architectural configuration. While at first it might seem that this is a simple task, we have found that component and version selection are subject to the obligations and rights stipulated with a component's associated IP license [ScA13c]. However, it is possible to determine, prior to integration, whether the subsequent candidate for release deployment may suffer from licensing problems or not. When conflicts or mis-matches are discovered, again prior to further build-time process actions, alternative components with the similar functional capabilities and interfaces but different licenses may be substituted. An integration and test build process can instantiate components into a reusable OA software product line design, as we can determine families of component version instances that can be substituted within the OA system. For example, Google Chrome browser in this configuration may replace the Firefox Web browser, because both are under permissive open source software (OSS) IP licenses.

Once an integrated is built from source code and linkage of pre-existing binary executables and connectors (e.g., dynamically loaded libraries) the many faces of testing the integrated assembly now begin. As testing is itself a reasonably complex process, we simply note that no single testing regime will guarantee the presence of faults, vulnerabilities, or bugs. All components should be subject to black box testing against known test cases, as well as self-defending cybersecurity mechanism and vulnerability testing. Detection of flaws may then necessitate white box testing, which requires access to source code, whose availability will be determined by component IP licenses and shared acquisition agreements. Integrated AC builds should be subjected to build test (aka, smoke test); regression tests; self-defending AC cybersecurity mechanism and vulnerability testing; and IP license compliance testing. All test results should also be organized as a test case suite aligned with reusable requirements, and stored within test data repository.

Some technical risks can be identified that pertain to the integration and packaging of components needed to build and test an AC include:

- Multi-component AC can be developed using different CI environments, tool chains, software code repositories, and software component file versions.
- Multi-component AC diversity can result in combinatorial large versions spaces that cannot be easily managed or tracked by people (e.g., acquisition program officers) without automated tools and support environments (e.g., configuration management systems tailored to produce acquisition support reports).
- Such diversity can, if manageable, enable new approaches to cybersecurity at integration build time that is not readily detectable or observable once the AC is deployed.

2.5 AC Release, Component Deployment, Installation and Local Configuration –

The software system you release and deploy depends on what (and how) you build and package for release and installation. However, what you build and what you release may not be the same, though they need to be functionally equivalent. For example, when you select one or more closed source software (CSS) components (an already compiled and integrated executable binary image) with a common restrictive IP license (i.e., one that prohibits copying or redistribution) for inclusion in a build-time AC configuration, during the build process, you must link it as an executable binary for inclusion in a release candidate for deployment (or deployment testing) on a local computer. Such inclusion is a prerequisite for overall integrated system testing processes required by CI. However, you cannot distribute such a release candidate to others, as it is common for CSS to not allow duplication or distribution of licensed copies of software binaries. Instead, we need to specify and configure a deployment-platform specific automated software installation mechanism (e.g., installation wizard) that needs to search for and find a local licensed copy of the CSS executable binary, and link it to the result of the build sub-process that provides a run-time linkage mechanism in expectation.

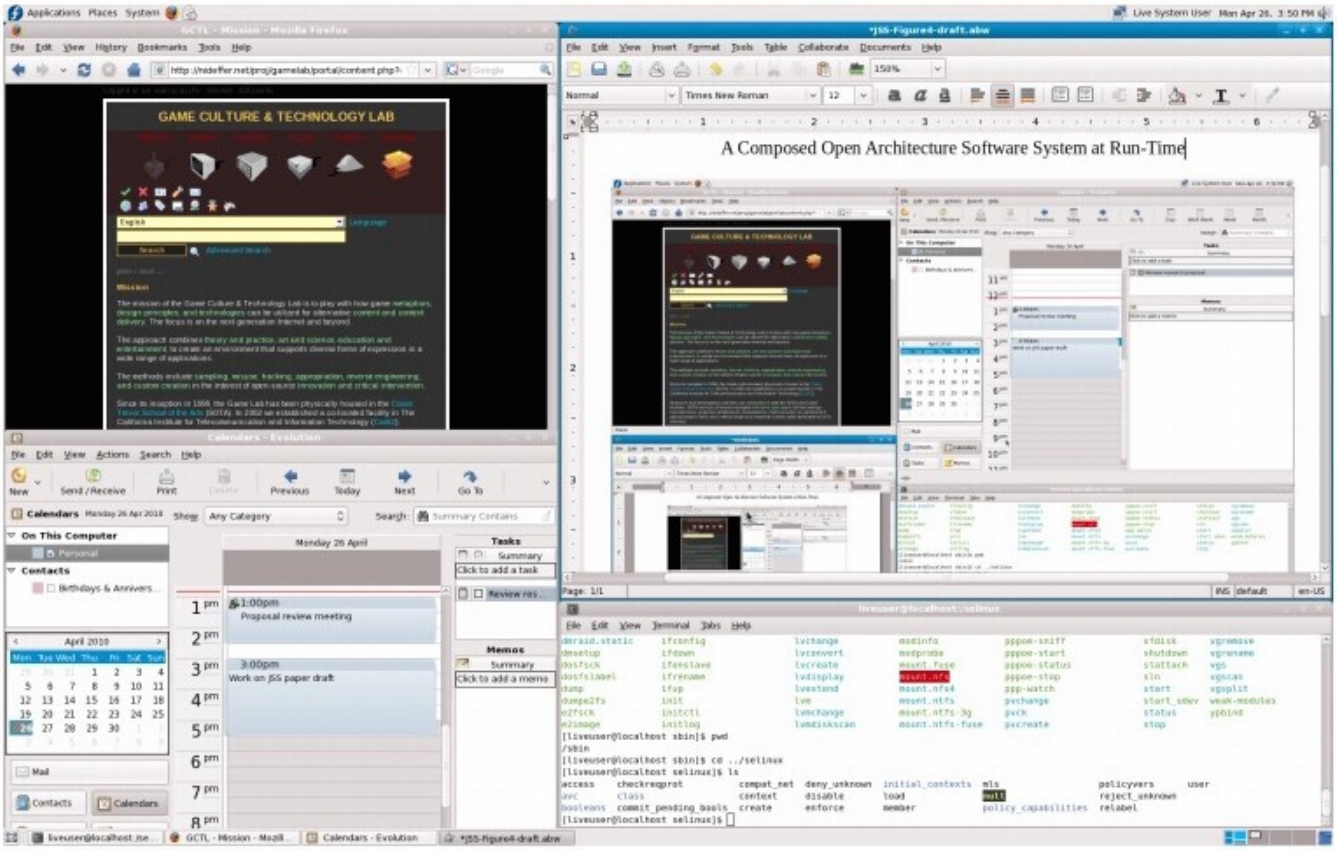


Figure 3. Screenshot of a PC desktop view of the released system seen in integration specification view in Figure 2, as well as compatible with the OA seen in Figure 1: *Firefox* Web browser upper left, *AbiWord* word processor upper right, *Evolution* email and calendaring app lower left, and the securable *SE Linux* operating system and command shell lower right.

A similar effort is needed to enable user acceptance testing or certification testing on their local platform. These AC release deployment process steps can be accomplished with some effort, but this effort could also be anticipated at design-time or build-time, when developers make their selection for which component instances to include in the AC integration and test build.

Some technical risks can be identified that pertain to the release, deployment and installation of components needed to install and configure the AC on end-user computing platforms include:

- Inability to detect remote end-user components or component versions that must be linked by an automated software installation utility, such that what is known to operate as intended at the system integrator site, does not operate as intended on the end-user platform.
- Component version mis-matches between integration and deployment platforms also applies to IP license versions associated with the components—component IP licenses must match or be compatible.
- Software installations may be partial due to processing delays or problems not visible to system integrators, and such installations need to be automatically verified by the installation utility, once resumed (or restarted) and completed.
- Some AC may be configured at build-and-test integration time with components that are not licensed for redistribution, so only a partial implementation of the AC is deployed and the end-user platform must provide the necessary mission components that can be linked to the partial AC to produce a fully operational AC.
- Component IP licenses must be carefully specified for the OA of an AC so that redistribution, local cybersecurity configuration, and local component IP license and version are known to be compatible in advance of deployment.
- Installed AC software configurations should be tracked and their as-built configuration specification serialized and stored in a repository at the integration site, to facilitate more rapid IP license and as-configured cybersecurity status, otherwise mis-matches will arise which are time-consuming to decipher and remedy.
- There is no prior software installation process that is known to be reliable without some sort of imposed common deployment platform (e.g., Apple iPhones), versus when the deployment platforms are subject to combinatorial large version spaces (e.g., Android-based smartphones prior to 2015).
- Shared agreements should account for software component/AC release installation processes and automated utilities that can verify what versions of what components with what IP licenses are installed and operational on a given end-user platform, and whether the installed Ac is configured appropriately for local cybersecurity protection, so as to minimize downtime resulting from problematic software update installations.

2.6 AC and Mission Component Post-deployment Adaptation and Evolution –

An OA system can evolve by a number of distinct mechanisms or process enactment pathways, some of which are common to all systems, but others of which arise only in OA

systems or where components in a single system are heterogeneously licensed [AAS10]. Figure 4 provides a summary of some of the various paths, as described elsewhere in greater detail [ScA12, ScA13c].

An OA system can evolve by a number of distinct mechanisms, some of which are common to all systems but others of which are a result of heterogeneous component licenses in a single system.

By component evolution— One or more components can evolve, altering the overall system's characteristics (for example, upgrading and replacing the *Firefox* Web browser from version 35 to 36). Such minor versions changes generally have no effect on AC system architecture.

By component replacement— One or more components may be replaced by others with modestly different functionality but similar interface, or with a different interface and the addition of shim code or scripting glue code to make it match.

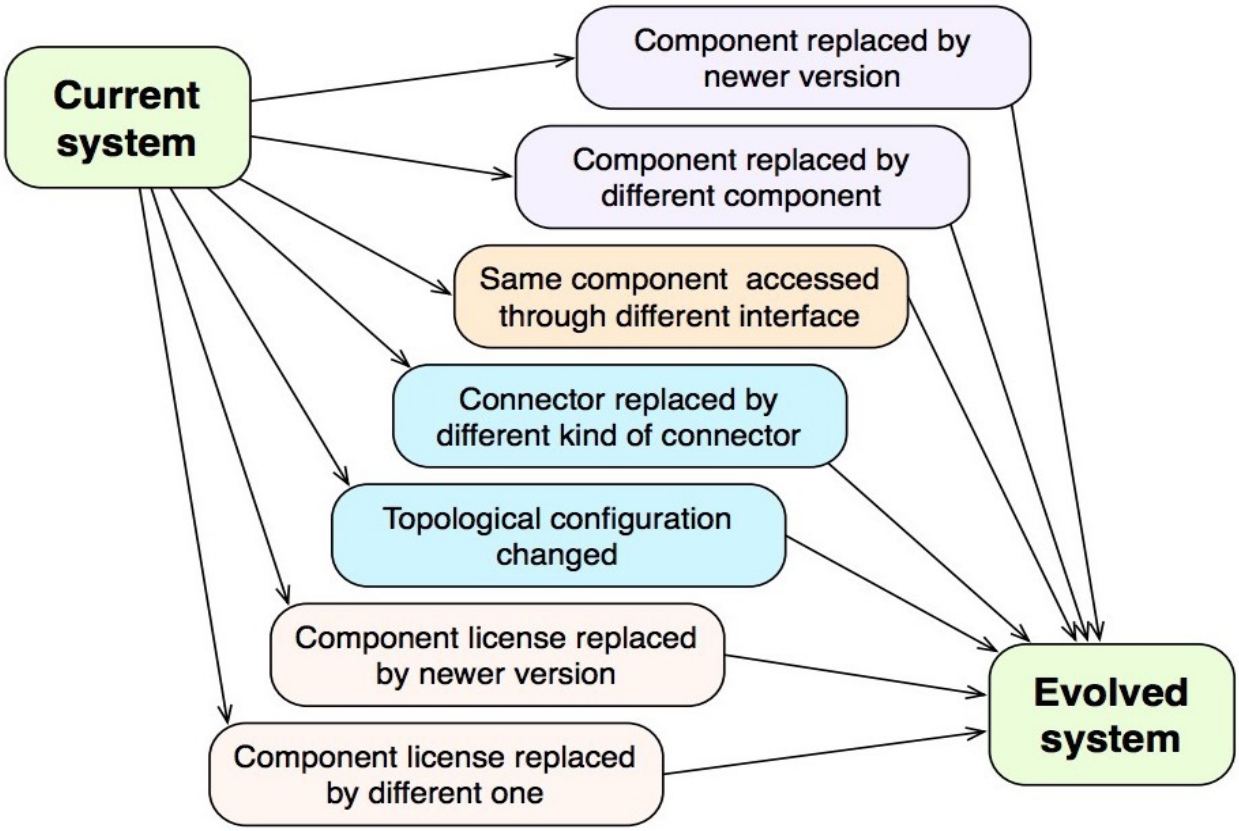


Figure 4. Ways and means for how composed systems or AC may evolve over time and technology.

By architecture evolution— The OA can evolve by changing connectors between components rearranging connectors in a different configuration, or changing the interface through which a connector accesses a component, altering the system characteristics. Revising or refactoring the configuration in which a component is connected can change how its license affects the

rights and obligations for the overall system. An example is the replacement of word processing, calendaring, email components, and connectors to them with Web-browser-based services such as Google Docs, Google Calendar, and Google Mail. Compare the end-user run-time version of the AC shown in Figure 5 to that previously seen in Figure 3, which is functionally similar, and is compatible with the OA design in Figure 1, yet has a different integration build architectural configuration. Thus cybersecurity attacks targeted to the OA system depicted in Figure 3 would not be effective against a functionally similar system displayed in Figure 5, due to architectural and data storage repository differences (e.g., local file system versus remote cloud services).

By component license evolution— The license under which a component is available may change. For example, IP license stipulations regarding technical data rights that were not available previously with earlier versions of this component are now available, due to a IP license update.

In response to different desired rights or acceptable obligations— The OA system's integrator or consumers may desire additional IP license rights (for example the right to sublicense and redistribute), or no longer desire specific rights; or the set of license obligations they find acceptable may change. In either case the OA system evolves, whether by changing components, evolving the architecture, or other means, to provide the desired rights within the scope of the acceptable obligations associated with one or more of the AC components' IP licenses.

Rapid dynamic system reconfiguration— More advanced evolution scenarios entail support for building and releasing of multi-variant AC deployment configurations that substitute functionally equivalent software component compilations (integration builds) that produce multiple, diverse executable binary images, each of which may execute in its own processor core, in a multi-threaded, multi-core processor. Rapid dynamic system reconfiguration may be part of an emerging cybersecurity strategy for realizing software diversification, whose goal is to reduce the attack surface on common AC configurations or components deployed on homogeneous run-time platforms. Pursuing this path requires a new compilation and build system regime that in turn anticipates a new generation of CI systems, repositories, and software installation utilities.

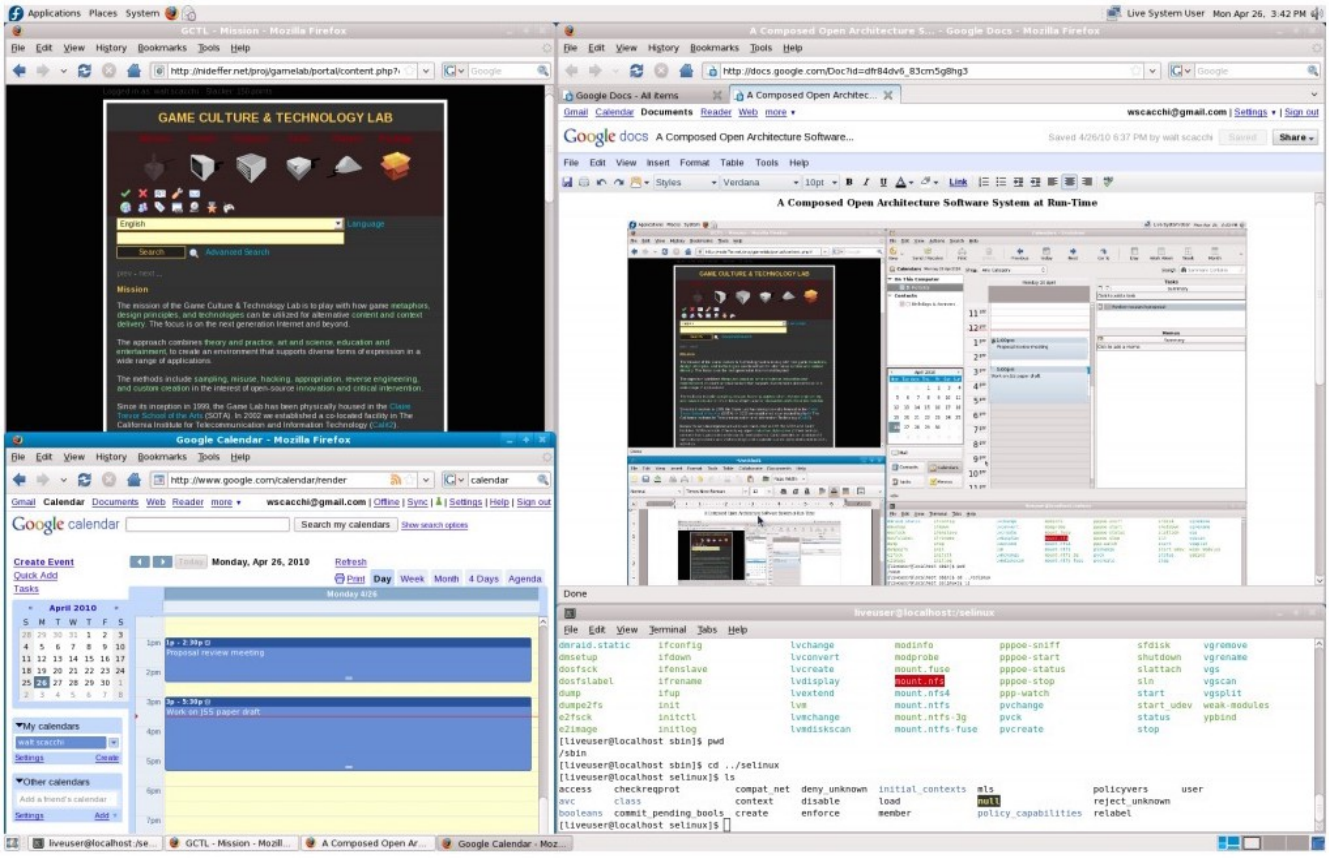


Figure 5. A screenshot view of a deployed end-user run-time configuration of the alternative OA system configuration (now using Web-based service apps) compared to that in Figure 3, resulting from system evolution.

Some technical risks can be identified that pertain to the post-deployment adaptation and evolution of components needed to sustain field operation of the AC include:

- Evolutionary changes to AC components, component licenses, or configuration may be additive, subtractive, or orthogonal; and also may or may not be upward compatible with the versions they replace.
- Non upward-compatible evolutionary changes do occur, especially when transitioning from components sourced by different producers or system integrators, or when large system development contractors acquire smaller component producers, or when large contractors acquire or merge with one another.
- End-users will need to become aware of the shifting functional performance and usage modes of evolving AC components, as well as how to work around them if component/AC functionality is degraded or denied.
- End-user modifications to deployed AC run-time configuration are possible or may be desirable to support rapid adaptation in the field. Such end-user customizations that create deployed AC variants whose architectural changes (e.g., component-connector interface modifications) need to be conveyed or remotely assessed for both added

value for sharing/redistribution and introduced vulnerabilities.

- Not updating deployed AC or components with known vulnerabilities (due to slow or policy-limited provisioning of software updates) means that end-users may know their AC may be compromised and thus less trustworthy.

4.0 Discussion and Conclusions

In our view, the best way forward is to invest is a focused R&D effort that can iteratively develop and deliver a domain-specific language (DSL) for specifying and modeling (1) the open architectures (OA) of assembled capabilities (AC) composed from mission software components and software interconnection mechanisms. Such a DSL can be (2) utilized across the AC development, deployment, and sustainment life cycle. Each life cycle stage either (3) creates, processes, analyzes, or updates the OA representation rendered in the DSL that is both (4) human-readable and computer processable. Furthermore, (5) the DSL can be utilized within an interactive environment that can be used by software producers, system integrators, customers, and acquisition authorities. Such a DSL-based interactive environment can be utilized in particular (6) to manage, track, and update specifications of functional requirements (FR) provided by missions components, sub-assemblies, and overall AC, as well as the (7) cybersecurity and (8) intellectual property (IP) licenses associated with different mission components, all within the DSL used to specify and annotate the OA for an AC. To be clear, this requires an investment in a focused R&D effort that can iteratively develop and deliver the DSL and supporting software as an open source, open architecture system, ideally subject to IP license that allows open access, modification and (possibly limited) redistribution, as well as support for specify cybersecurity requirements processing capabilities.

Overall, these notes are meant to help surface our starting assumptions that then help frame life cycle activities that arise during the rapid acquisition of legacy, custom, or reusable mission components and Assembled Capabilities supporting C3CB systems. AC for C3CB systems will be acquired across an ecosystem of multiple parties: component producers, system integrators and diverse customers/end-users. Choices made regarding the employment of Open Architectures and reusable AC Reference Models that are explicitly rendered in human-readable and computer-processable domain-specific languages, along with automated tools and techniques for modeling and analyzing appear key to enabling the rapid acquisition of adaptable mission components and agile integrated AC.

Comments or questions regarding any of the materials in these notes are welcome by the authors. We look forward to your requests for revision or simplification where appropriate, as well as to requests for further information on the concepts described in these notes.

Acknowledgements

Preparation of this report is supported by grant #N002444-15-1-0010 from the Acquisition Research Program at the Naval Postgraduate School, Monterey, CA. No endorsement, review, or approval implied. This paper reflects the views and opinions of the authors, and not necessarily the views or positions of any other persons, group, enterprise, or government agency.

References

[AAS10] Alspaugh, T.A., Asuncion, H. and Scacchi, W. (2010). Software Licenses in Context: The Challenge of Heterogeneously Licensed Systems, *Journal of the Association for Information Systems*, 11(11), 730-755, November 2010.

[AlS13] Alspaugh, T.A., and Scacchi, W. (2013). Ongoing Software Development Without Classical Requirements, *Proc. 21st. IEEE Intern. Conf. Requirements Engineering*, Rio de Janeiro, Brazil, 165-174, 15-19 July 2013.

[DBA15] Diercks, P., Brockman, B., and Ansari,J. (2015). *Secure Web Integration Framework*, Slide presentation, ACWG/BMTT Working Group Meeting, 3 June 2015. Also see, Galdorisi, G., Brockman, B., Diercks, P., George, A., et al. (2014). Achieving Information Dominance: Unleashing the Ozone Widget Framework, *Proc. 19th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper 109, Alexandria, VA, June 2014.

[GGM14] George, A., Galdorisi, G., Morris, M. and O'Neil (2014). DoD Application Store: Enabling C2 Agility. *Proc. 19th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-104, Fairfax, VA, June 2014.

[GMH13] George, A., Morris, M., Galdorisi, G., Raney, C., Bowers, A., and Yetman, C. (2013) Mission Composable C3 in DIL Information Environments using Widgets and App Stores. *Proc. 18th Intern. Command and Control Research and Technology Symposium*, Paper-036, Alexandria, VA, June 2013.

[GMO14] George, A., Morris, M. and O'Neil, M. (2014). Pushing a Big Rock Up a Steep Hill:

Lessons Learned from DoD Applications Storefront, *Proc. 11th Annual Acquisition Research Symposium*, Vol. 1, 306-317, Naval Postgraduate School, Monterey, CA.

[GSS15] Guertin, N.H., Sweeney, R., and Schmidt, D.C. (2015). How the Navy Can Use Open Systems Architecture to Revolutionize Capability Acquisition: The Naval OSA Strategy

Can Yield Multiple Benefits. *Proc 12th Annual Acquisition Research Symposium, Monterey, CA,* NPS-AM-15-004, May 2015.

[Ken15] Kendall, F. (2015). *Implementation Directive for Better Buying Power 3.0*, http://www.acq.osd.mil/fo/docs/betterBuyingPower3.0%289Apr15%29.pdf

[MaS12] Mactal, R., Spruill, N. (2012). A Framework for Reuse in the DoN. *Proc. 9th Acquisition Research Symposium,* Vol.1, 149-164, Naval Postgraduate School, Monterey, CA.

[ScA12] Scacchi, W. and Alspaugh, T.A. (2012). Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *Journal of Systems and Software*, 85(7), 1479-1494, July 2012.

[ScA13a] Scacchi, W. and Alspaugh, T.A. (2013a). Advances in the Acquisition of Secure Systems Based on Open Architectures, in *Journal of Cybersecurity & Information Systems,* 1(2), 2-16, February 2013.

[ScA13b] Scacchi, W. and Alspaugh, T.A. (2013b). Streamlining the Process of Acquiring

Secure Open Architecture Software Systems, *Proc. 10th Annual Acquisition Research Symposium,* Monterey, CA, 608-623, May 2013.

[ScA13c] Scacchi, W. and Alspaugh, T.A. (2013c). Processes in Securing Open Architecture Software Systems, *Proc. 2013 Intern. Conf. Software and System Processes*, 126-135, May 2013, San Francisco, CA.

[ScA13d] Scacchi, W. and Alspaugh, T.A. (2013d). Challenges in the Development and Evolution of Secure Open Architecture Command and Control Systems, *Proc. 18th Intern. Command and Control Research and Technology Symposium*, Paper 098, Alexandria, VA, June 2013.

[ScA14b] Scacchi, W. and Alspaugh, T. (2014). *Cost-Sensitive Acquisition of Open Architecture Software Systems for Mobile Devices*, Invited Presentation, MITRE-ATARC Workshop on Challenges in Legal and Acquisition, Federal Mobile Computing Summit, Washington, DC, 19 August 2014.

[ScA14c] Scacchi, W. and Alspaugh, T. (2014). *Reasoning about the Security of Open Architecture Software Systems for Mobile Devices*, Invited Presentation, Federal Mobile Computing Summit, Washington, DC, 20 August 2014.

[ScA15] Scacchi, W. and Alspaugh, T.A. (2015). Achieving Better Buying Power Through Acquisition of Open Architecture Software Systems for Web-Based and Mobile Devices, Proc. 12th Annual Acquisition Research Symposium, pgs. 4-21, Monterey, CA, May 2015.

[WoS11] Womble, B., Schmidt, W., Arendt, M., and Fain, T. (2011). Delivering Savings with Open Architecture and Product Lines, *Proc. 8th Acquisition Research Symposium*, Vol. 1, 8-13, Naval Postgraduate School, Monterey, CA.

Chapter 5:

Starting Assumptions on Life Cycle Activities for Acquiring Software-Based Assembled Capabilities

Starting Assumptions on Life Cycle Activities for Acquiring Software-Based Assembled Capabilities

Walt Scacchi and Thomas Alspaugh
Institute for Software Research
University of California, Irvine
Irvine, CA 92697-3455 USA
wscacchi@gmail.com

Version of: 28 July 2015

1.0 Introduction

In this note, we identify and briefly describe starting assumptions for the acquisition life cycle of assembled capabilities (AC) for software-intensive command, control, communications, cyber, and business systems (C3CB). Such systems may incorporate widgets, plug-ins, apps, and other mission components that accommodate Web-based or mobile devices.

Our overall goal is to work towards a new approach to acquisition of AC that is timely, produces technologically superior solutions, and does so in a cost-effective, agile, and adaptive manner. Said differently, our overall goal is to identify how best to provide new ways and means for streamlining AC acquisition through timely, agile, and adaptive technologically superior ways and means.

2.0 Starting Assumptions

Our starting assumptions for the goals of acquiring AC into four groups, account for Acquisition, AC, IP and Cybersecurity requirements, and technical support requirements. The assumptions thus include:

Acquisition

1. Rapid acquisition life cycles are measured in months instead of years (e.g., 2-12 months) through ways and means that are compliant with relevant Federal Acquisition Regulations.

2. We seek to enable new rapid acquisition life cycle practices best suited for producing adaptive, agile, AC within multi-party ecosystems of mission component producers, integrators/assemblers, and customers/end-users.

3. Acquisition includes the processes, support systems, personnel, review and approval activities, etc. associated with the procurement, development, deployment, and field support of assembled capabilities or their components.

4. In November 2013, the Department of Defense (DoD) released Interim DoD Instruction 5000.02, "Operation of the Defense Acquisition System." Enclosure 2 of this document describes policies that apply to the management of acquisition programs. Regulations require that programs develop an intellectual property (IP) strategy as part of the acquisition strategy.

Assembled Capabilities (AC)

5. In concept, AC incorporate four kinds of software or data components, identified here.

Some of these components may represent *new elements* to be developed via acquisition, or may be already deployed *legacy systems* that may need to be refactored and componentized for use within the AC via acquisition.

5.1 *Mission* specific widgets, plug-ins, mashups or applications ("apps") possibly acquired via an online app store;

5.2 *Common technology* (e.g., Ozone widget framework, Java); and

5.3 *Shared infrastructure* (Joint Information Environment, MPE, ICITE).

5.4 *Data components* (sensor signal processing, data aggregation/reduction, data repositories, reusable source code libraries, etc.) are collectively composed/composable through custom capability-specific integration code (e.g., software scripting or glue code).

31. Data components may also incorporate business logic elements, which may themselves be treated as business logic components.

6. The range of components within an AC can include:

6.1 Standalone programs ("apps");

6.2 Executable binary code;

6.3 Run-time common technology libraries, frameworks, or middleware;

6.4 Inter-app (glue code) and intra-app (widget) scripting code;

6.5 Configured systems or existing sub-systems, such as for infrastructure services.

7. Software or data components used in AC are composed/assembled using software-based *connectors* as interconnection mechanisms. Different connectors may utilize mission specific hardware or network technologies within the AC. However, apps that are hardware-unique should be avoided, as this limits their potential reuse, and instead should follow the Joint C2 Reference Architecture to insure their portability. Different types of software/data connectors include:

7.1 source code that accesses application program interfaces (APIs) or Web-service interfaces;

7.2 reusable source code or re-entrant executable libraries;

7.3 data messaging protocols (e.g., specific to a network);

7.4 middleware (e.g., Mobile Device Manager);

7.5 new/legacy databases;

7.6 network file systems; etc.

8. The Joint C2 Reference Architecture provides a solid basis for an AC approach that maximizes decoupling between applications through the use of well-defined service interface exposure and service usage.

9. AC employ explicit, open architectures (OAs) that specify how components are connected together through specific interconnection mechanisms:

9.1 The goals of AC with OA are to improve adaptation and transparency, accommodate innovative mission components, increase competition, and other initiatives aligned with OUSD

(AT&L) *Better Buying Power* (BBP) guidelines [Ken15].

9.2 OAs may conform to explicit (open source) reference models of known, viable, and reusable capabilities, cybersecurity requirements, or acquisition processes.

9.3 OA references models should be specified and represented in explicit, open source formats that can be shared, modified, and redistributed within a community of interest.

9.4 Reusable OAs or reference models may be created and shared to support common AC development scenarios.

10. AC must accommodate mobile and Web-based C3CB platforms subject to cybersecurity requirements and assurance processes.

11. AC may be assembled or field-adapted by system integrators that can include external contractors, internal consultants, Program Offices, deployed forces personnel, or others in the Mission Partner environment, where appropriate.

12. AC may incorporate components, sub-assemblies, or assembly acquired/provided by one or more Program offices, subject to shared (acquisition) agreements.

13. AC can be verified for their cybersecurity status, and resistance to known vulnerabilities and cyber attack scenarios, in light of asynchronous evolution of AC mission components.

Assembled Capabilities Intellectual Property and Cybersecurity Issues

14. Software components are subject to intellectual property (IP) licenses determined by their producers. IP licenses stipulate obligations (e.g., covenants like fee paid per user licenses) that end-user organizations/participants, as well as system integrators (capability assemblers), must satisfy in order to realize rights associated with the use, modification, or redistribution of components within assembled capabilities:

14.1 In November 2013, the Department of Defense (DoD) released Interim DoD Instruction 5000.02*, Operation of the Defense Acquisition System*. Enclosure 2 of the document describes policies that apply to the management of acquisition programs. Regulations require that programs *develop an IP strategy* as part of the acquisition strategy [SEI13].

14.2 IP licenses may be permissive and provide open source *computer software* and/or *technical data* rights, or be restrictive and closed (proprietary) with limited computer software and/or technical data rights for customers/end-users.

14.2.1 Technical data "means recorded information, regardless of the form or method of the recording, of a scientific or technical nature (including computer software documentation). The term does not include computer software or data incidental to contract administration, such as financial or management information" (DFARS 252.227.7013 (15)).

14.2.2 Computer software "means computer programs, source code, source code listings, object code listings, design details, algorithms, processes, flow charts, formulae and related material that would enable the software to be reproduced, recreated, or recompiled. Computer software does not include computer databases or computer software documentation" (DFARS 252.227.7013 (a)(3)).

14.3 Permissive and restrictive licenses fall along a spectrum of possibilities, from most

permissive (least restrictive) to most restrictive (least permissive), as determined by their specified obligations to be fulfilled in order to realize the rights available.

15. Software interconnection mechanisms (connectors) may or may not be subject to proprietary IP licenses or “fair use” doctrine (cf. Oracle v. Google, re: Java APIs, now pending before the SCOTUS):

15.1 Use of closed source or proprietary software interconnection mechanisms mitigate against use of OA.

15.2 Software interconnection mechanisms may or may not transfer IP license obligations and rights between components, or across the AC, depending on their design, implementation and usage within an AC.

15.3 Systems or AC with closed or unknown architectures may or may not transfer IP licenses or cybersecurity requirements across interconnected components, and may be subject to the inversion or denial of rights due to evolutionary changes in included system components.

15.4 In Europe (and thus much of NATO), the European Copyright Directive expressly excludes APIs from copyright.

16. Program Offices and other acquisition service providers will need to specify and assess the IP obligations they are willing to accept, and what rights they require/desire, in the components and assembled capabilities they seek to acquire, deploy, and adapt.

16.1 System integrators (capability assemblers) determine which IP obligation and rights are realized in an AC delivered to a customer organization or end-users, through the integrators choices when configuring the AC using specified interconnection mechanisms and components.

16.2 Program offices or other acquisition service providers need to provide guidance to system integrators and verify compliance with IP obligation stipulations in order to insure end-user and technical data rights.

17. Shared agreements for acquiring AC across Program offices/agencies are either a kind of:

17.1 “Unilateral agreement” where a single party is chartered to produce, maintain and operate a set of components or services which are intended for re-use by third parties without requiring shared agreements between parties. In this case, a Service Level Declaration provides the basis for establishing the expectation for the components' functionality, performance and availability.

17.2 “Bilateral agreement” between two parties to share acquisition costs, IP obligations and rights, and cybersecurity assurance.

17.3 “Multi-lateral agreement” among three or more parties to share acquisition costs, IP obligations and rights, and cybersecurity assurance.

17.4 Shared agreements must be iteratively negotiated in advance of use.

17.4.1 Worked examples of shared agreements would be beneficial for review and training

17.4.2 Worked examples should include identification of representative trade-off alternatives

among the parties.

17.5 Shared agreements should be reusable and tailorable across Program Offices

17.5.1 Federal Acquisition Regulation (and DFAR) compliance should be assured with constraints allowing variability within the terms and conditions for reuse and tailoring.

17.5.2 Shared agreements that are reusable and tailorable provide pre-certified FAR/DFAR compliance will streamline acquisition of AC covered by such agreements.

Technical System Requirements Going Forward

18. Cybersecurity requirements also entail obligations (e.g., mandatory access control) and rights (e.g., authorized end-users can access sensor data), so that the tools, techniques, and processes for assuring IP license compliance are also usable for cybersecurity requirements assurance.

19. IP and cybersecurity obligations and rights for mission components and AC can be expressed in human-readable and computer-processable formats, such as through use of a domain-specific language (DSL) designed for such purposes a) IP and cybersecurity obligations and rights can be used to annotate the components interfaces or sub-assembly interfaces within an OA AC, using such a DSL.

20. Such a DSL may also explicitly model the OA for specific AC, and for tailorable product lines of AC [GSS15, MaS12, ScA12, WoS11].

20.1 Product lines correspond to, and are specified with, reference models, as reference models specify a family of systems or AC that share a common OA and functionally similar components.

20.2 Product lines are an established approach that accommodates reuse and tailoring of product line family members.

20.3 Many types of software components and connectors for AC may be grouped into product lines, where family members are either functionally equivalent, functionally similar, or distinct.

20.4 Shared agreements may also be grouped into product lines.

21. Such a DSL enables and streamlines acquisition of AC through use of an open source software-based system for managing, analyzing, and tracking AC OA, IP and cybersecurity obligations and rights.

22. Such a DSL can serve as the basis for automated calculation of metrics regarding AC development status, AC architectural completion gaps and component interface mis-matches, IP obligations (like license costs), extent of new versus reusable requirements within or across AC product lines, frequency and ease of post-deployment adaptation into AC, and more.

Though this is quite a laundry list of what is required to acquire AC, it does help to provide a context for identifying the issues and opportunities that can arise during the acquisition of AC for C3CB. Conversely, failure to recognize or explicitly declare such starting assumptions will

necessitate the need to restate and recapitulate the problems encountered in order to realize the breakthrough solutions sought for acquiring AC needed for emerging missions that utilize new, innovative technologies and platforms. Warfighters and supporting military forces will increasingly be treated as active C3CB elements. This will be especially true in the near future when warfighters and supporting forces are adorned with body-worn sensors and hand-held/mobile devices that are networked with command centers, combat platforms/vehicles, logistics supply chains, and personnel readiness assessment capabilities, potentially across joint forces.

3.0 Conclusions

Technological superiority is a common goal, as is streamlining AC acquisition. So starting assumptions matter, as do visions for emerging technological opportunities, risks, and vulnerabilities. This note thus serves to collect the assumptions we are making as we seek to move towards articulating the acquisition life cycle for AC and mission components that are rapid, agile, and adaptive across an ecosystem of component producers, system integrators or capability assemblers, and customers/end-users. Please help us identify any missing assumption, or to suggest revisions that will help improve the value and quality of this note.

Acknowledgements

Preparation of this report is supported by grant #N002444-15-1-0010 from the Acquisition Research Program at the Naval Postgraduate School, Monterey, CA. No endorsement, review, or approval implied. This paper reflects the views and opinions of the authors, and not necessarily the views or positions of any other persons, group, enterprise, or government agency.

References

[GSS15] Guertin, N.H., Sweeney, R., and Schmidt, D.C. (2015). How the Navy Can Use Open Systems Architecture to Revolutionize Capability Acquisition: The Naval OSA Strategy

Can Yield Multiple Benefits. *Proc. 12th Annual Acquisition Research Symposium, Monterey, CA,* NPS-AM-15-004, May 2015.

[Ken15] Kendall, F. (2015). *Implementation Directive for Better Buying Power 3.0*, http://www.acq.osd.mil/fo/docs/betterBuyingPower3.0%289Apr15%29.pdf

[MaS12] Mactal, R., Spruill, N. (2012). A Framework for Reuse in the DoN. *Proc. 9th Acquisition Research Symposium,* Vol.1, 149-164, Naval Postgraduate School, Monterey, CA.

[ScA12] Scacchi, W. and Alspaugh, T.A. (2012). Understanding the Role of Licenses and Evolution in Open Architecture Software Ecosystems, *Journal of Systems and Software*, 85(7), 1479-1494, July 2012.

[SEI13] Software Engineering Institute (2013). Managing Intellectual Property in the Acquisition of Software Systems—SPRUCE/SEI, November.

[WoS11] Womble, B., Schmidt, W., Arendt, M., and Fain, T. (2011). Delivering Savings with Open Architecture and Product Lines, *Proc. 8thAcquisition Research Symposium*, Vol. 1, 8-13, Naval Postgraduate School, Monterey, CA.

Chapter 6:

Tutorial Presentation: Beyond Open Architecture: Issues, Challenges, and Opportunities in Open Source Software Development (OSSD) for Aerospace and Defense Applications

Beyond Open Architecture: Issues, Challenges, and Opportunities in Open Source Software Development (OSSD) for Aerospace and Defense Applications

Walt Scacchi

Overview

- Background
- Sample of research findings on OSSD
- OSSD as multi-project software ecosystems
- OSSD, open architectures, and software licenses
- Challenges in securing OA C2 systems
  - *Case Study*: Securing the development and evolution of an OA C2 system within an agile, adaptive software ecosystem
- Emerging transformations with OSS and OA systems
- Conclusions

Background

Personal History of OSS Development Studies

- 2000-2015 (60+ publications)
  - Computer games, defense, X-ray astronomy, Internet/Web infrastructure, bioinformatics, higher education, e-commerce, neuroscience, virtual reality.
- *Discovering* requirements practices and processes across OSS communities of practice.
- Participant *role sets*, *role migration*, and *social movements* within/across OSS projects.
- Open architecture (OA) systems with *heterogeneously licensed* components.

What is open source?

- Open source software (OSS) denotes specifications, representations, socio-technical processes, and multi-party coordination mechanisms in *human readable, computer processable* formats.
- Socio-technical control of OSS is elastic, negotiated, and amenable to decentralization.
- OSS development subsidized by contributors and participants.

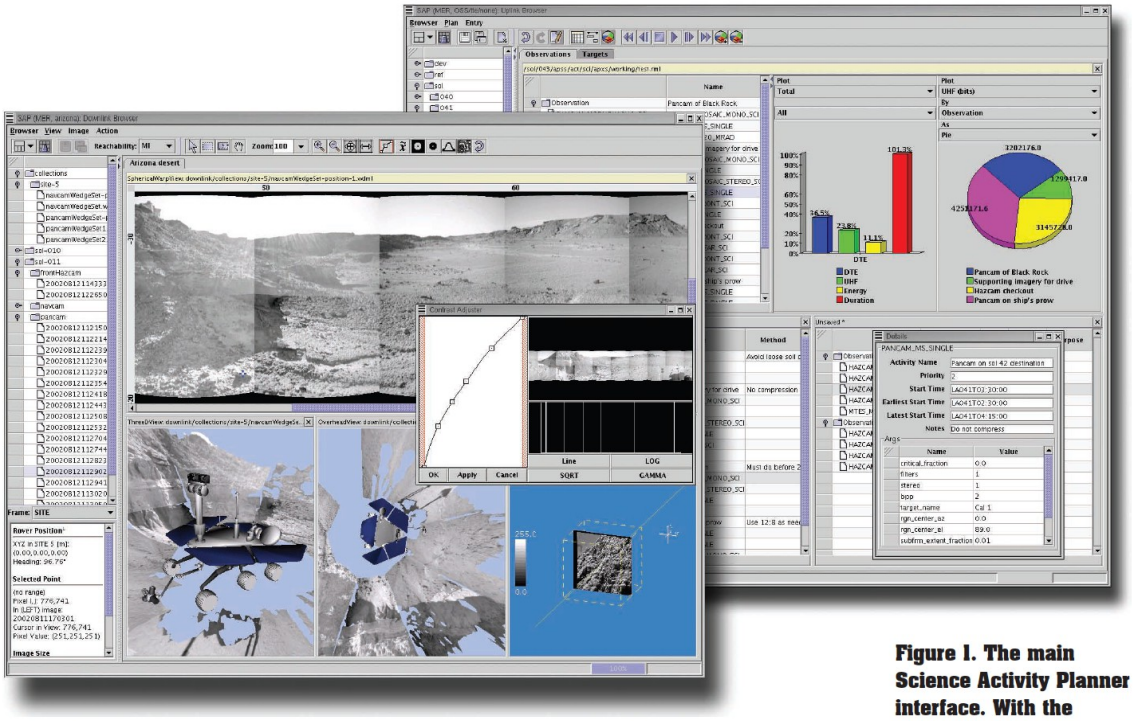
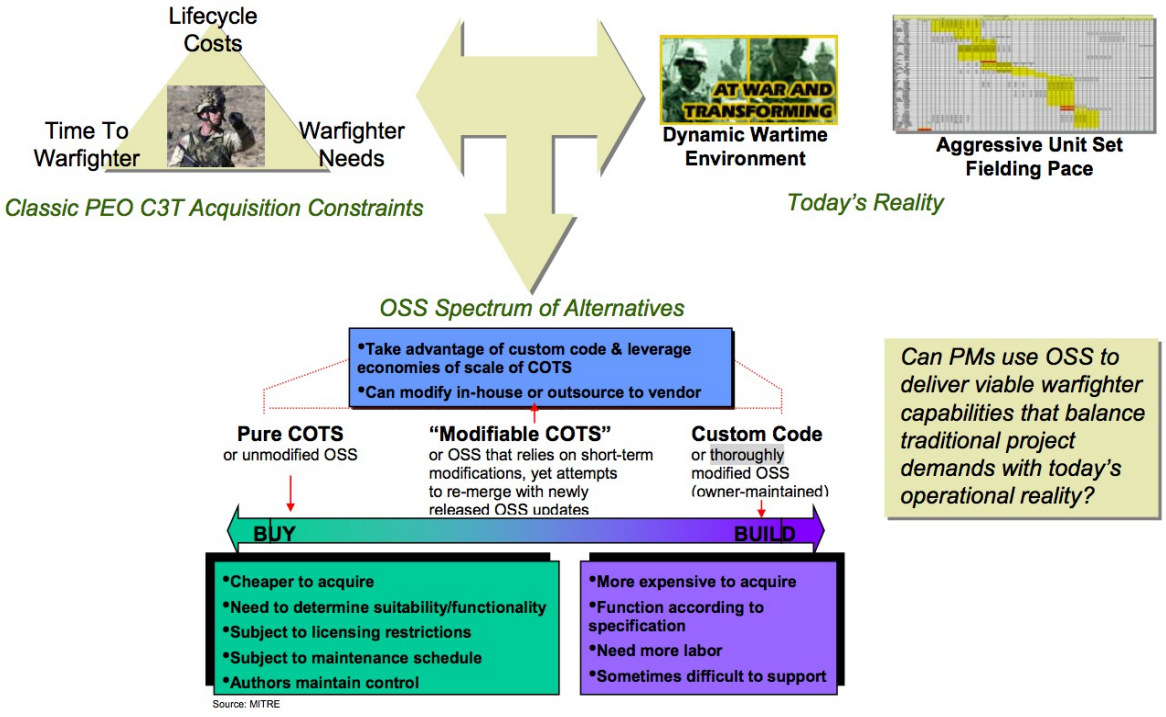


Figure 1. The main Science Activity Planner interface. With the downlink browser

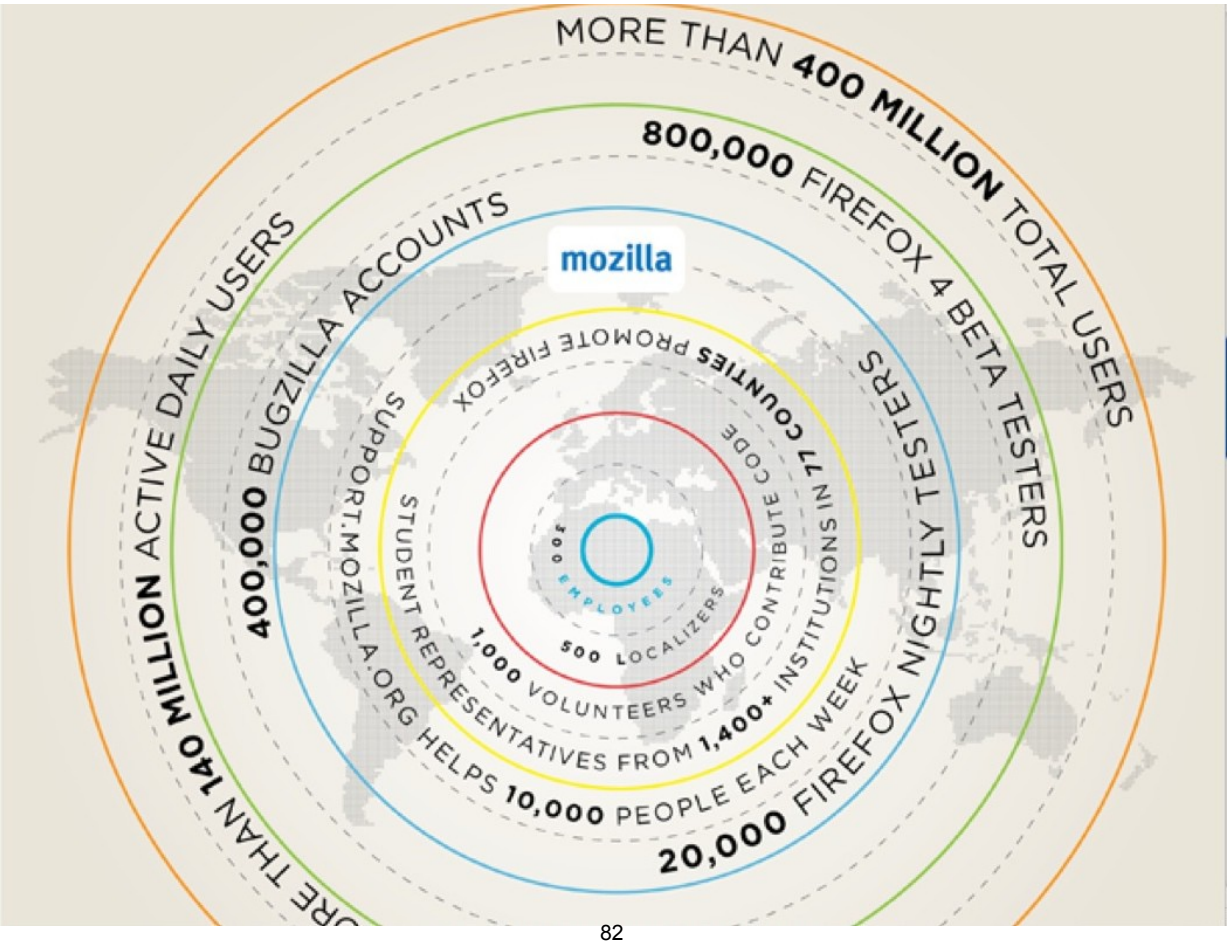
Source: J.S. Norris, Mission Critical Development of Open Source Software: Lessons Learned, *IEEE Software*, 21(1), 42-49, Jan 2004.



Sample of research findings on OSSD

What is free/open source software development?

- Free (as in “freedom” or liberty, not cost) vs. open source
  - Freedom to access, browse/view, study, modify and redistribute the source code—it's about *IP licenses*!
  - Free is always open, but open source is not always free
- FOSSD is not “software engineering”
  - *Different*: FOSSD can be faster, better, and cheaper than SE in some circumstances
  - FOSSD teams use 10-500+ OSSD tools (versions) and communications applications to support their development work



LibreOffice
The Document Foundation

Home of the LibreOffice Productivity Suite

Home | Download | Features | Get Help | Get Involved | Developers | **International Sites** | About Us | Security

International Sites

LibreOffice has a number of teams working on localization into different languages. For software, support and documentation in your preferred language, please check below to see if we currently have a site serving your locale. If you don't see a site in the language you are looking for, please consider joining our community and getting involved in working to fill the gap.

http://ar.libreoffice.org/	العربية	Arabic
http://bo.libreoffice.org/	རྫོང་ཁ	Tibetan
http://cs.libreoffice.org/	čeština	Czech
http://da.libreoffice.org/	Dansk	Danish
http://de.libreoffice.org/	Deutsch	German
http://el.libreoffice.org/	Ελληνικά	Greek
http://eo.libreoffice.org/	Esperanto	Esperanto
http://es.libreoffice.org/	Español	Spanish
http://et.libreoffice.org/	Eesti keel	Estonian
http://fa.libreoffice.org/	فارسی	Persian
http://fi.libreoffice.org/	Suomi	Finnish
http://fr.libreoffice.org/	Français	French
http://gl.libreoffice.org/	Galego	Galician
http://he.libreoffice.org/	עברית	Hebrew
http://hi.libreoffice.org/	हिन्दी	Hindi
http://hu.libreoffice.org/	Magyar	Hungarian
http://it.libreoffice.org/	Italiano	Italian
http://ja.libreoffice.org/	日本語	Japanese
http://lo.libreoffice.org/	□□□□□□□	Lao
http://lt.libreoffice.org/	Lietuvių kalba	Lithuanian

sourceforge | Search | Browse | Enterprise | Blog | Jobs | Deals | Help

SOLUTION CENTERS | Go Parallel | Resources | Newsletters

Most downloads over all time

Rank	Project Name	Downloads
1	Microsoft's TrueType core fonts	1.4B
2	Notepad++ Plugin Manager	993.6M
3	VLC media player	897.1M
4	eMule	682.1M
5	Vuze - Azureus	542.1M
6	MinGW - Minimalist GNU for Windows	435.2M
7	FileZilla	413.6M
8	7-Zip	409.1M
9	Ares Galaxy	406.5M
10	PortableApps.com: Portable Software/USB	361.5M

Most downloads last week

Rank	Project Name	Downloads
1	Microsoft's TrueType core fonts	13.1M
2	egbucket	8.0M
3	Notepad++ Plugin Manager	7.4M
4	MinGW - Minimalist GNU for Windows	2.0M
5	Apache OpenOffice	1.0M
6	FileZilla	966.1K
7	PortableApps.com: Portable Software/USB	961.5K
8	Ubuntuzilla: Mozilla Software Installer	624.1K
9	bucket	537.5K
10	Scrollout F1	405.4K

OSSD Enterprise Models

- Free Software (GPL)
- Permissive Open Source (BSD/MIT, FreeBSD)
- Corporate/Inner Source (Hewlett-Packard)
- Consortium/Alliance (OSDL, SugarCRM)
- Non-profit foundations (Apache, Mozilla, Gnome, Perl)
- Corporate-Sponsored (Google, HP, IBM, Microsoft, Oracle)
- Open Modding Extensions to Closed Source (many game companies)
- Community Source (Sakai, Westwood)

---------- *not* OSSD models below ----------

- Shared Source via Non-Disclosure Agreement
- Open Systems (open APIs, closed components)

OSSD Project Characteristics

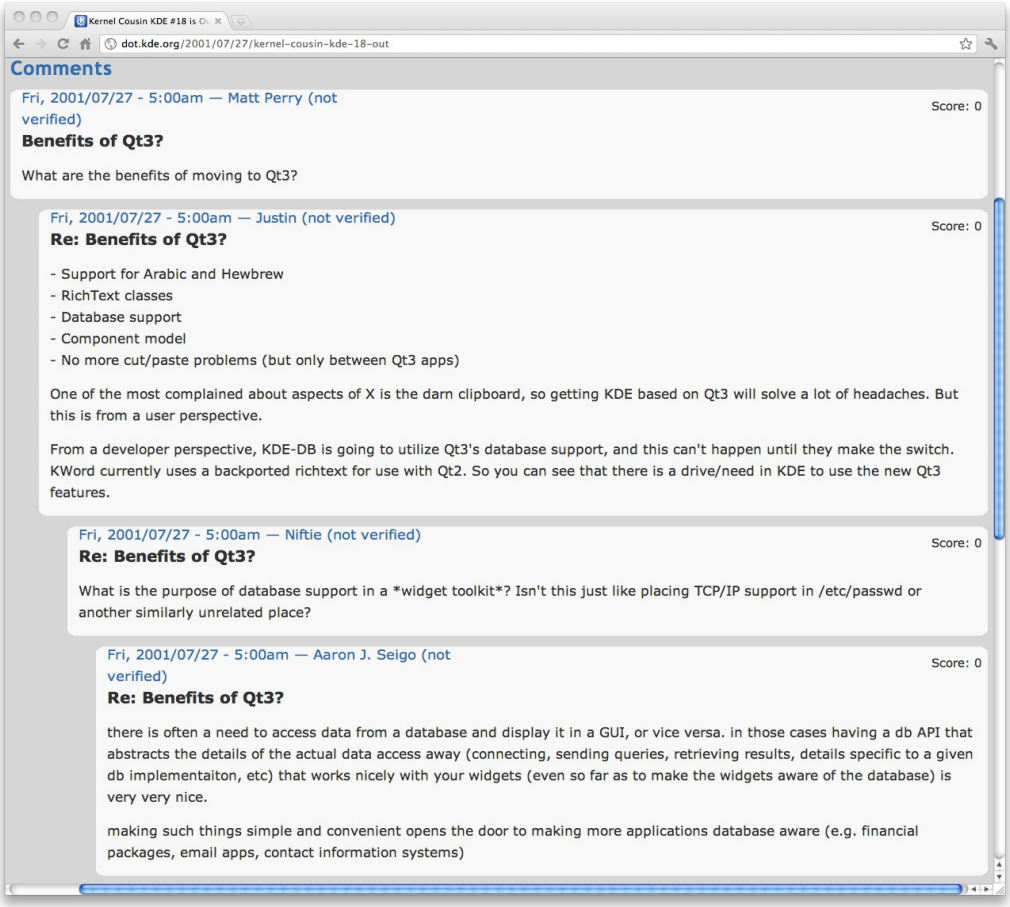
- Operational code early and often--actively improved and continuously adapted
  - Short-cycle (FOSS) vs. long-cycle (SLC) time processes
- *Post-facto* software system requirements and design
  - FOSSD has its own “-ilities” which differ from those for SE
- *Caution*: the vast majority (>90%) of OSSD projects fail to grow or to produce a viable, sustained software release.

OSSD Project Characteristics

- FOSS developers are typically *end-users* of what they build
  - Some FOSS users (~1%) are also FOSS developers
  - These developers know their "functional requirements"
- Requires *critical mass* of contributors and FOSS components connected through socio-technical interaction networks
  - Can be <1% of user community for large FOSS projects
  - Sustained commitment and contribution critical
- OSSD projects can emerge/evolve via *bricolage*
  - Unanticipated architectural (de)compositions
  - Multi-project component integrations

OSSD Informalisms

- Software *informalisms*--artifacts participants use to describe, proscribe, or prescribe what's happening in a project
- Informalisms capture detailed rationale and debates for what changes were made in particular development activities, artifacts, or source code files

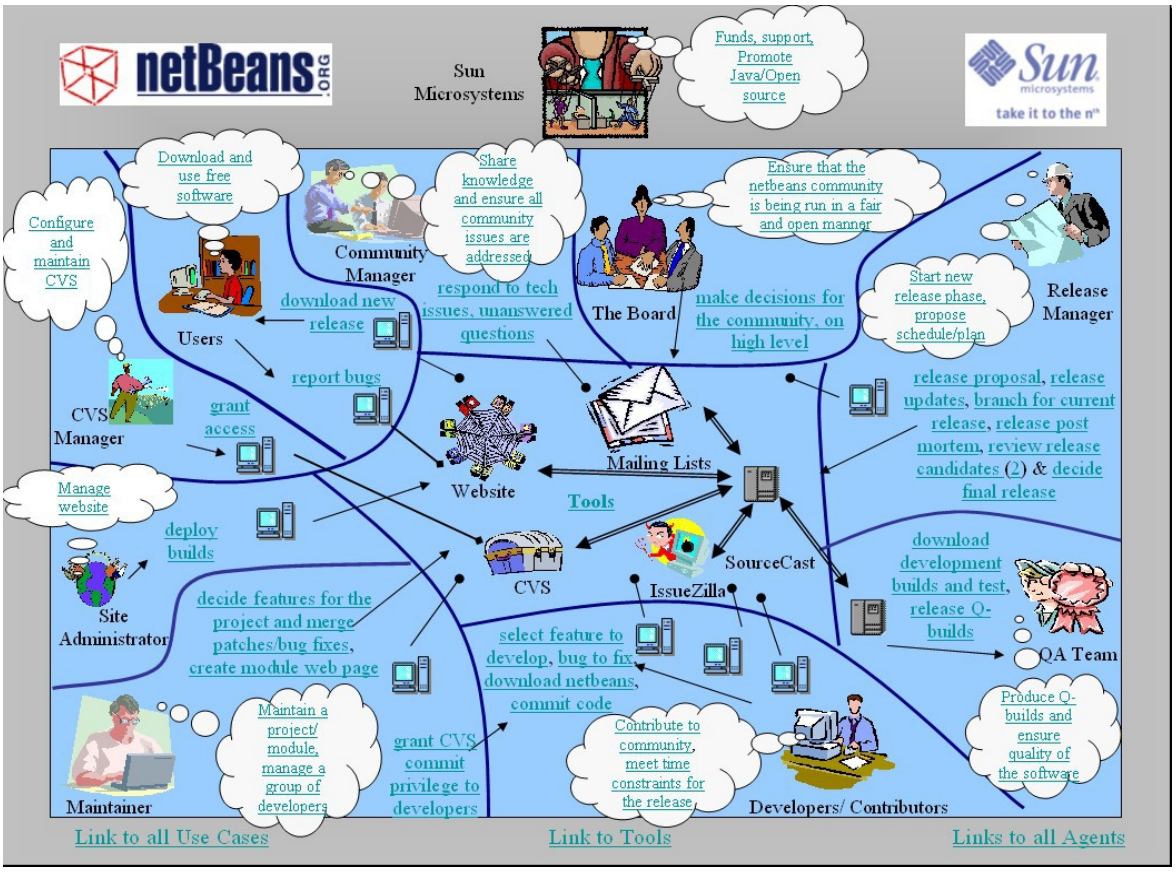


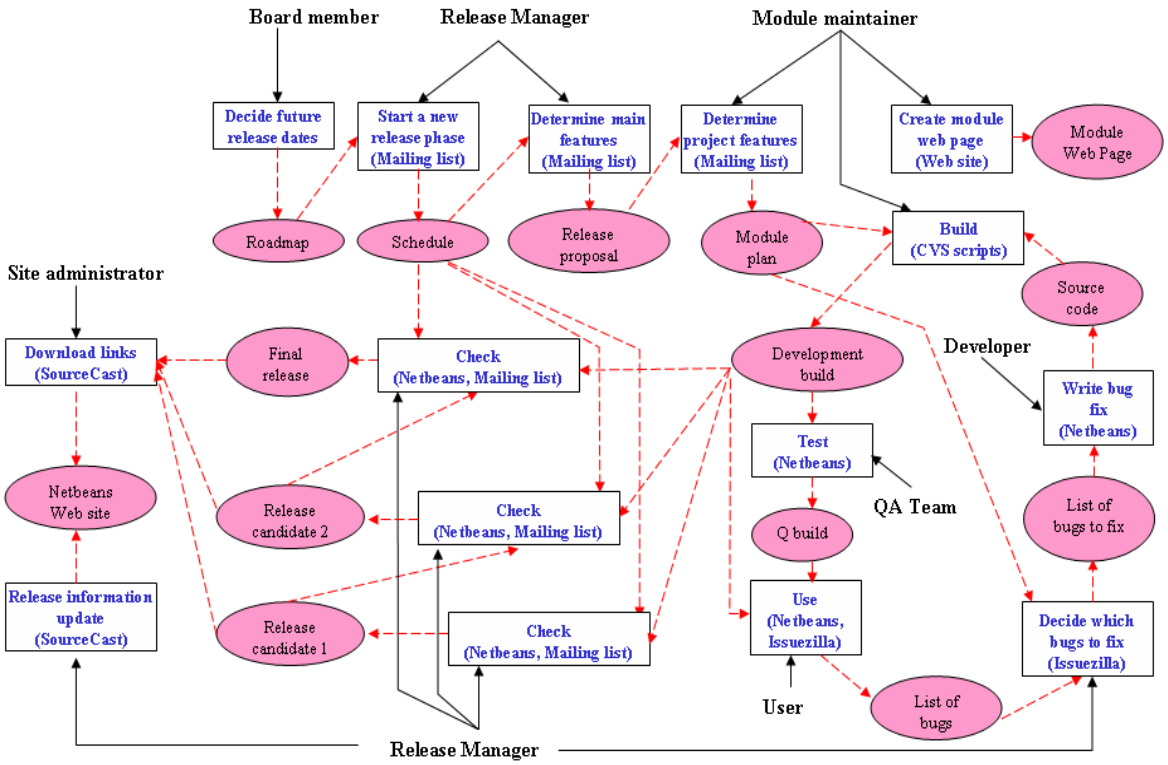
OSSD *Informalisms: Artifacts* and *repositories* enable collaboration in OSS development

Email lists	Discussion forum	News postings	Project digests
IM/Internet Relay Chat	Scenarios of usage	How-to guides	Screenshots
FAQs; to-do lists: item lists	Project Wikis	System documentation	External publications
Copyright licenses	Architecture diagrams	Intra-app scripting	Plug-ins
Code from other projects	Project Web site	Multi-project portals	Project source code
Project repositories	Software bug reports	Issue tracking databases	Blogs, videos, social media.

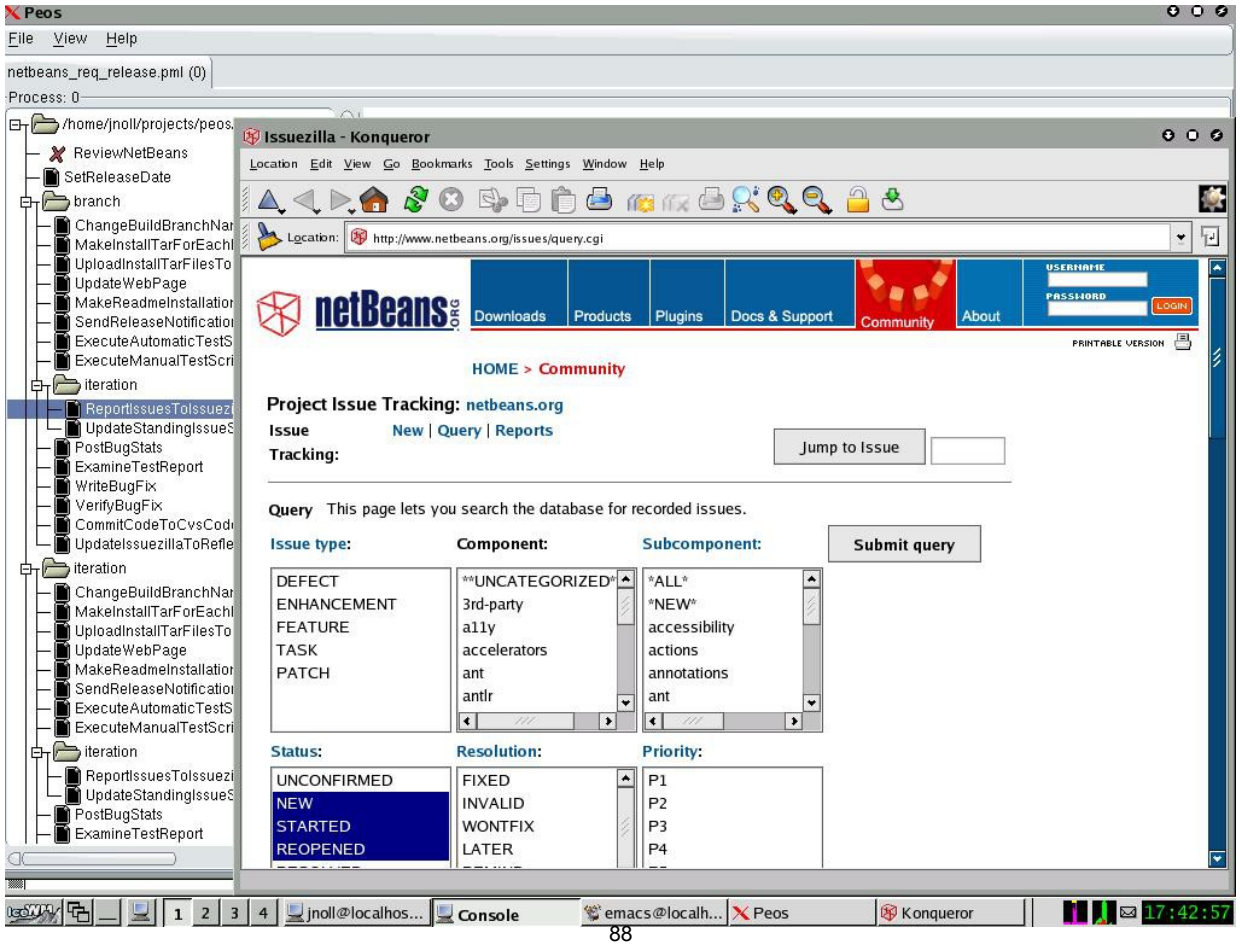
OSSD Processes?

- How does OSSD occur in practice, or via prescription?
- Who does what, where, when, how, why?
- OSSD projects in the wild eschew explicit models of their processes, preferring that contributors “learn” how things get done, by whom, etc. as a way to demonstrate your commitment to project success.
- Models of OSSD processes are few and mostly scarce, yet should be key to industrial OSSD utilization.
  - *Informal*, *formal*, and *executable* models desirable.





Legend: Boxes are *activities* (using *informalisms*); Ellipses are *resources* required or provided; Actor *roles* in boldface; *flow dependencies* as arrows.



OSSD Research Surveys

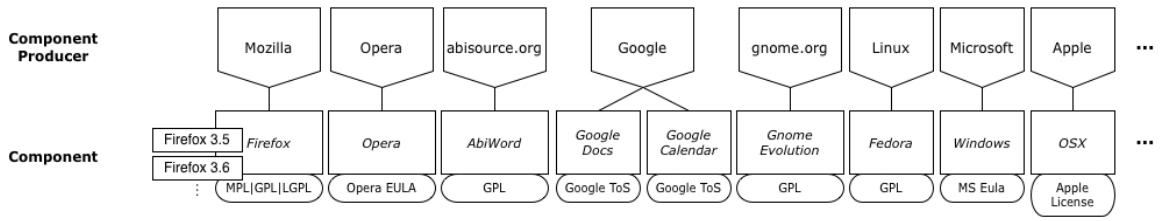
- Scacchi, W. (2007). Free/Open Source Software Development: Recent Research Results and Emerging Opportunities. *Proc. 6th. ESEC/FSE*, 459–468. Also see, Scacchi, W. Free/Open Source Software Development: Recent Research Results and Methods, in M.V. Zelkowitz (ed.), *Advances in Computers*, 69, 243-295, 2007.
- Gasser, L. and Scacchi, W. (2008). Towards a Global Research Infrastructure for Multidisciplinary Study of Free/Open Source Software Development, in *Open Source Development, Community and Quality*; B. Russo, E. Damiani, S. Hissan, B. Lundell, and G. Succi (Eds.), IFIP Vol. 275, Springer, Boston, MA. 143-158.
- Hauge, O., Ayala, C. and Conradi, R. (2010). Adoption of Open Source Software in Software-Intensive Organizations - A Systematic Literature Review. *Information and Software Technology,* 52(11), 1133-1154.
- Aksulu, A. and Wade, M.R. (2010). A Comprehensive Review and Synthesis of Open Source Research, *J. Assoc. Info. Systems*, 11(11), 576-656.
- Scacchi, W., Crowston, K., Jensen, C., Madey, G., Squire, M., and others (2010). *Towards a Science of Open Source Systems*, Final Report, Computing Community Consortium, November 2010. http://foss2010.isr.uci.edu/content/foss-2010-reports/
- Höst, M., Oručević-Alagić, A. (2011). A Systematic Review of Open Source Software in Commercial Software Product Development, *Information and Software Technology*, 53(6), June, 616-624.
- Crowston, K., Wei, K., Howison, J., and Wiggins, A. (2012). Free/libre open source software development: what we know and what we do not know. *ACM Computing Surveys*, 44(2).

OSSD as multi-project software ecosystems

What is a (software) ecosystem?

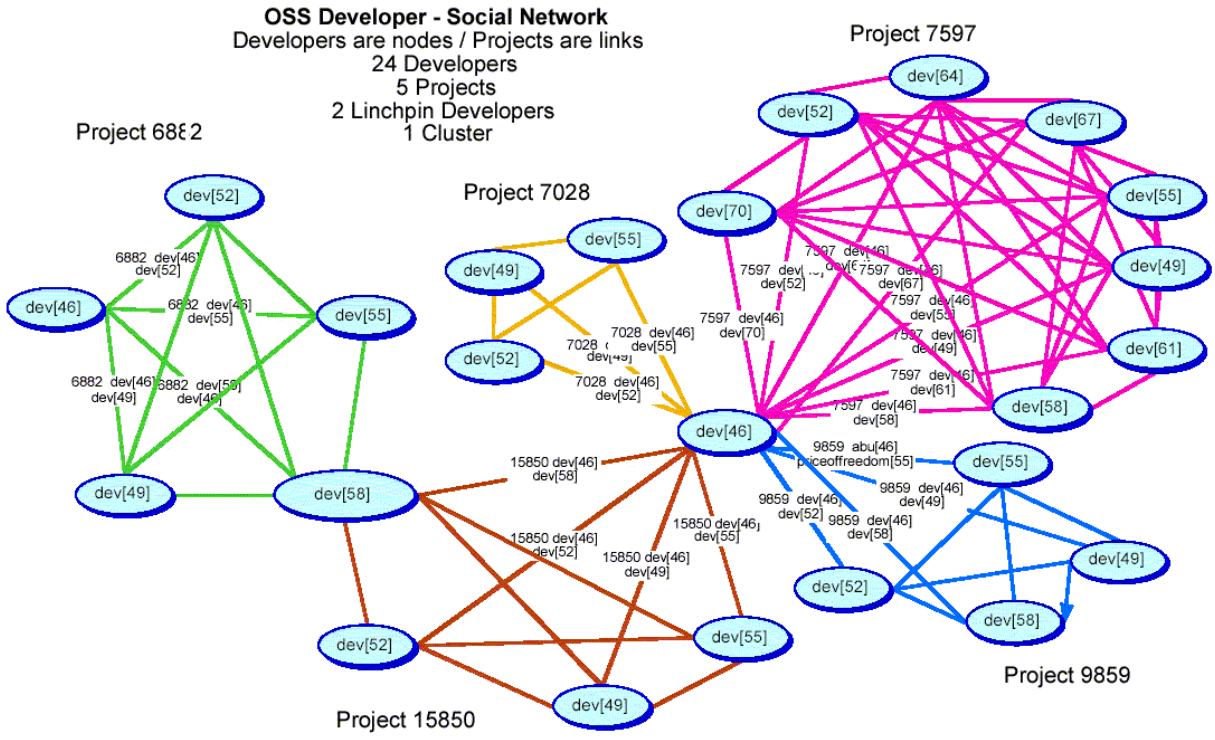
- An ecology of systems with diverse species juxtaposed in adaptive prey-predator food chain relationships.
- Economic network of processes that transform the flow of resources, enacted by actors in different roles, using tools, to produce products, services, or capabilities.
- *Software supply network* of component producers, system integrators, and consumers.

Software ecosystem of producers and the software components for an enterprise or Web-compatible C2 system

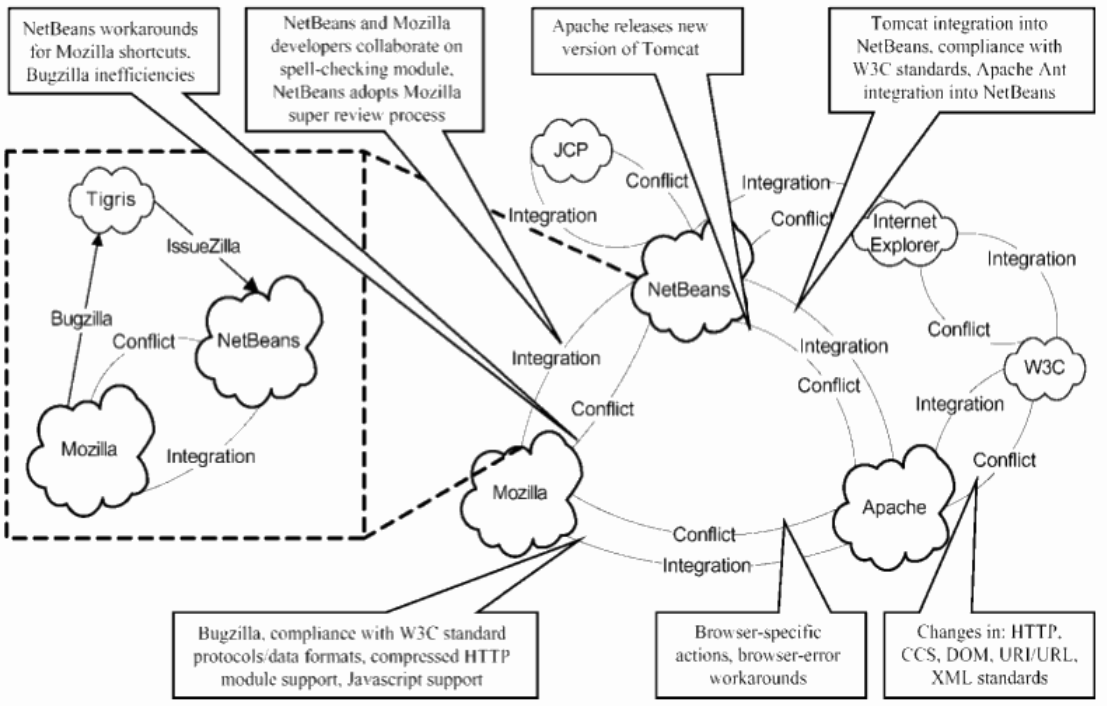


Multi-project software ecosystem

- Mutually dependent OSS development and evolution propagate architectural styles, dependencies, and vulnerabilities
- *Architectural bricolage* arises when autonomous OSSD projects, artifacts, tools, and systems co-mingle or merge
  - Enables discontinuous or *exponential growth* of FOSS code, functionality, complexity, contributions



Source: G. Madey, *et al.,* 2005



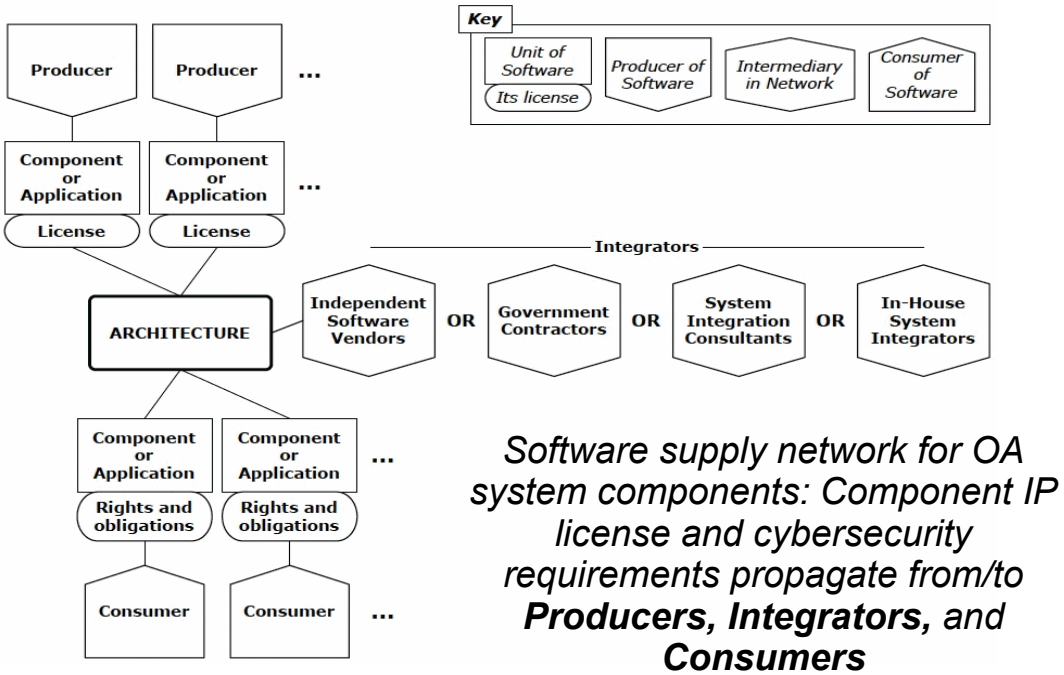
Source: C. Jensen and W. Scacchi, Process Modeling Across the Web Information Infrastructure, *Software Process--Improvement and Practice*, 10(3), 255-272, July-September 2005.

OSSD, open architectures, and software licenses for C2 or C3CB systems

What is an Open Architecture?

- DoD has announced policies and initiatives that commit to the acquisition of software-intensive systems that require or utilize an Open Architecture, and Open Technology.
- OA systems may include components with open APIs, OSS technology or development processes.
- Air Force, Army, and Navy each have their own reasons for adoption OA systems.
  - But what happens when there are conflicts across the services regarding what an OA is?
- Therefore, is it clear what an OA is?

OA software ecosystems

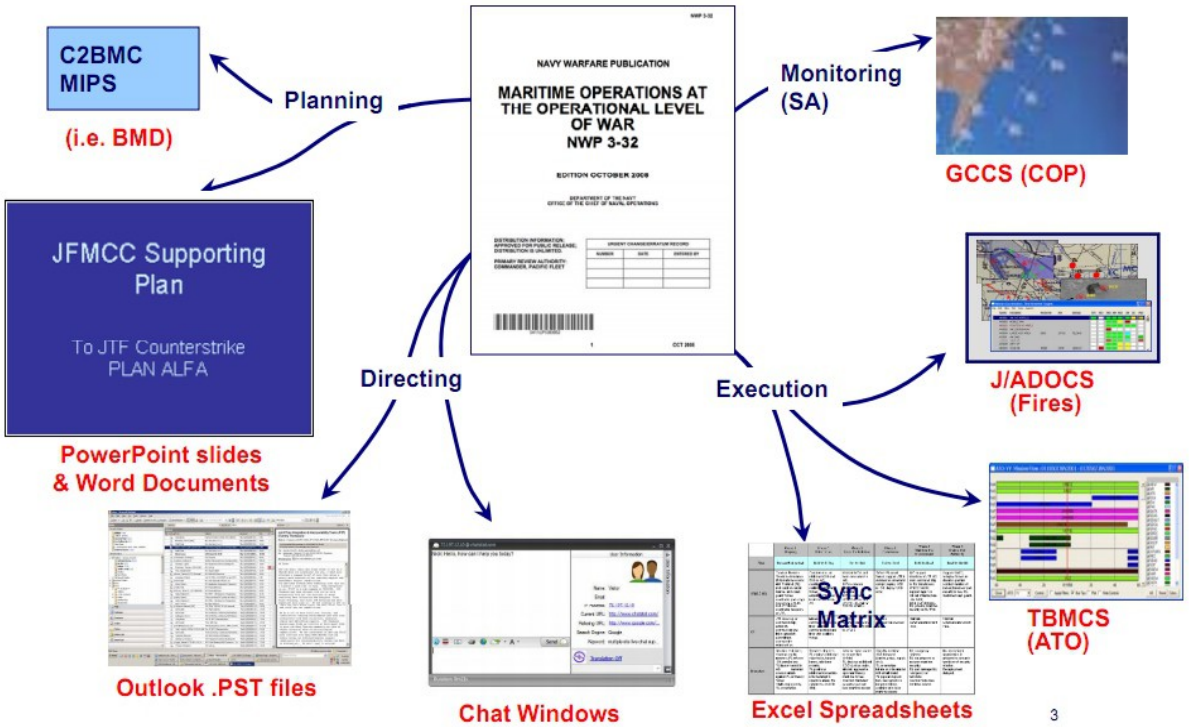


Software supply network for OA system components: Component IP license and cybersecurity requirements propagate from/to ***Producers, Integrators,*** *and* ***Consumers***

What is a C2 or C3CB System?

- C2 – Command and Control System
- C3CB – Command, Control, Communications (C3), Cyber and Business System
- Common features
  - Centralized architecture (decentralized in R&D)
  - Applications include: word processing, presentation, spreadsheet, email, calendar, messing, mapping, visualization/media display, resource scheduling, global shared displays, personal workstations, special devices.
  - Deployments in military operations, space operations, public utilities, network operations, live media (sports events) broadcasting, gaming centers, motorsports (*F1*).

Recurring vision for advanced C2: C2RPC [2010]



Decentralized Command and Control (DCC)

- DCC emerging as a new strategic thrust
  - [DoD JOAC 2012]
- DCC encourages physically decentralized user practices, using low-cost/open source software.
- DCC operates as a *virtual enterprise*
  - Physically distributed, logically centralized
  - "Edge" of a multi-site organization [Albert and Hayes 2003]
- DCC accommodates peer-to-peer organizational decision-making and work locations
  - Crowd-sourced DCC may also be possible

DECENT: a prototype platform for research and experimentation with DCC

- DECENT is an online virtual world prototype and platform for experimentation with DCC
  - Used in early studies of C2 mission planning games
- Developed with low-cost, open source software for virtual worlds (OpenSim).
  - *Transformative*: Potential to dramatically reduce the cost of fielding C2 system capabilities
  - Few barriers to acquisition
  - Applicable to mission planning and coordination in physical and virtual applications (Cyberwarfare)

Scacchi, W. Brown, C. and Nies, K. (2012).
Exploring the Potential of Virtual Worlds for Decentralized Command and Control, *Proc. 17th Intern. Command and Control Research & Technology Symposium*, Paper-096, Alexandria, VA.

Virtual worlds and physical places for C2

- Virtual worlds (VWs) can be used to *mirror* physical spaces, actors (avatars), devices, activities and resources within them.
- DECENT models physical C2 worlds and many mission planning IT resources as found in C2RPC [2010] for web-based C2.
- DECENT integrates remote text, chat, speech, image and video stream servers/clients
- DECENT supports decentralized, networked users
  - C2 activities with physically distributed users, within logically centralized/shared VW

Physical C2 facility

Virtual world for C2: *DECENT*

Under-explored topics for DECENT

- Most VW software technologies, including *OpenSim* and *Second Life* offer little/no ready support for *integration of external application programs or other software components*.
- *Securing a VW for military C2 applications* is a major concern in advancing this line of research and deployment.
- Availability of OSS+VW platforms supports low-cost R&D of next-gen C2/C3CB systems.

Conclusions and recommendations for studies for DECENT approaches to C2

- DECENT demonstrates a transformative reduction in cost of rapidly creating and deploying C2 systems supporting DCC via OSS
- DECENT can be deployed via pocket storage devices (flash storage thumb drives)
- Much remains to be studied using DECENT like technologies and approaches to DCC
- R&D should seek to demonstrate future benefits and articulate system/software risks (e.g., security of VWs is an open problem).

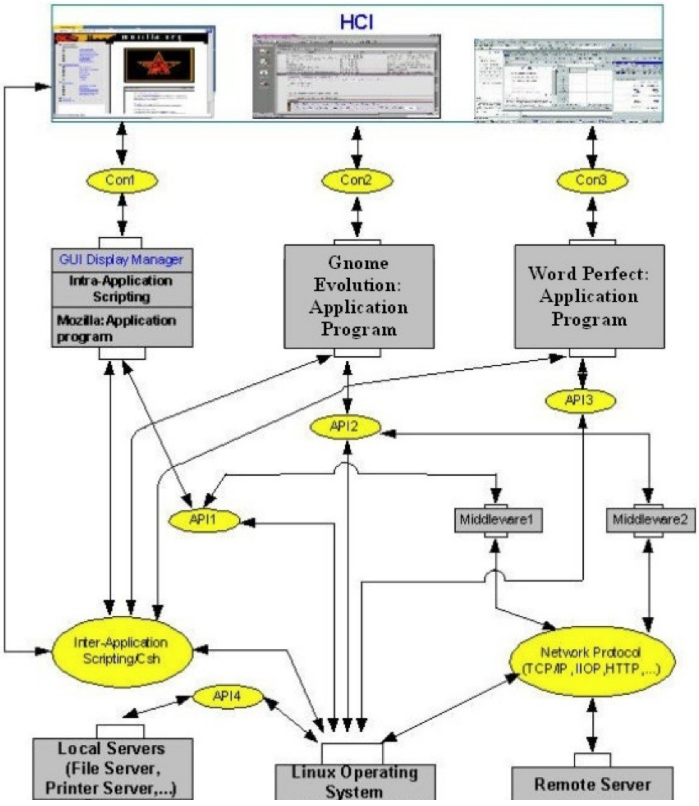
Open Architectures, OSS, and software license analysis

- *Goal*: identify software architecture principles and OSS licenses that mediate OA
- OSS elements subject to different IP licenses
- DoD policies and initiatives encouraging OA with OSS elements
- How to determine the requirements needed to realize OA strategies with OSS?

Source: W. Scacchi and T. Alspaugh, Emerging Issues in the Acquisition of Open Source Software within the U.S. Department of Defense, *Proc. 5th Annual Acquisition Research Symposium*, Vol. 1, 230-244, NPS-AM-08-036, Naval Postgraduate School, Monterey, CA, 2008.

Open (Software) Architecture Concepts

- Software source code components
  - Standalone programs
  - Libraries, frameworks, or middleware
  - Inter-application script code (e.g., for mash-ups)
  - Intra-application script code (e.g., for Rich Internet Apps.)
- Executable software components (binaries)
- Application program interfaces (APIs)
- Software connectors
- Configured sub-system or system



Legend: Grey boxes are *components;* ellipses are *connectors;* white boxes are *interfaces*; arrows are data or control *flow paths*; complete figure is architectural design *configuration*

OSS elements subject to different *IP licenses*

- Intellectual Property (IP) licenses stipulate obligations (requirements) and rights (capabilities) regarding use of the IP
  - GPL (Gnu Public License) stipulate right to access, study, modify, and *reciprocal* obligation to redistribute modified source
  - Mozilla now offers a "tri-license" for its software like Firefox:
    - GPL, MPL (lightweight), or Restricted (accommodating proprietary services)
  - Other OSS covered by different rights and obligations
- How to determine which *IP* obligations and rights apply to a configured system?
  - At *design-time* (maximum flexibility)
  - At *build-time (*may/not be able to redistribute components at hand)
  - At *run-time* (may/not need to install/link-to components from other sources)
  - At *evolution-time* (component suppliers, licenses, connections, etc. may change).

Source: Scacchi W. and Alspaugh, T. (2012). Understanding the Role of Licenses and Evolution in Open Architecture Software Ecoystems, *Journal of Systems and Software*, 85(7), 1479-1494, July.

OSS elements subject to different *Security licenses*

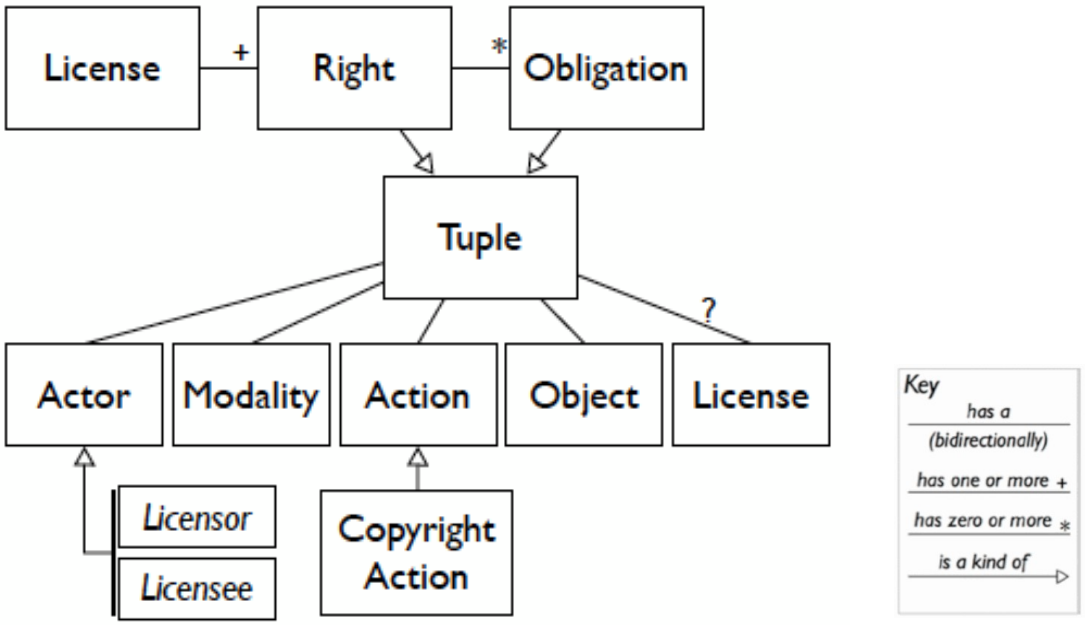
- Software system security obligations and rights are comparable to IP!
  - Security licenses can serve similar purposes (and analysis) as that for IP licenses.
- How to determine which *security* obligations and rights apply to a configured system?
  - At *design-time* (maximum flexibility)
  - At *build-time (*may/not be able to redistribute components at hand)
  - At *run-time* (may/not need to install/link-to components from other sources)
  - At *evolution-time* (component suppliers, licenses, connections, etc. may change).

Source: Scacchi, W. and Alspaugh, T. (2013). Advances in the Acquisition of Secure Systems Based on Open Architectures, *Journal of Cybersecurity & Information Systems*, 1(2), 2-16, February 2013.

Specifying and analyzing system access control requirements as “licenses”

- Security policies imply capabilities that correspond to “rights and obligations” in licenses
- Should be possible to specify and analyze system *security architecture* that conform to a *security meta-model*, much like we do for software licenses
- Should be possible to develop computational tools and development environments that can analyze security at design-time, build-time, and run-time, as well as when the system evolves

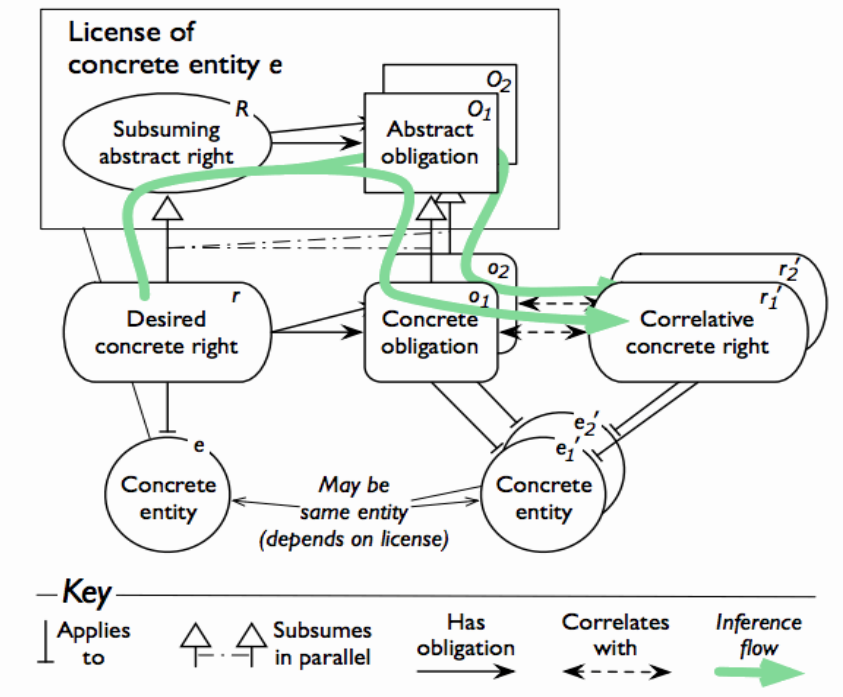
Software license meta-model for specifying constraint annotations



Logical modality and objects of software license rights and obligations constraints

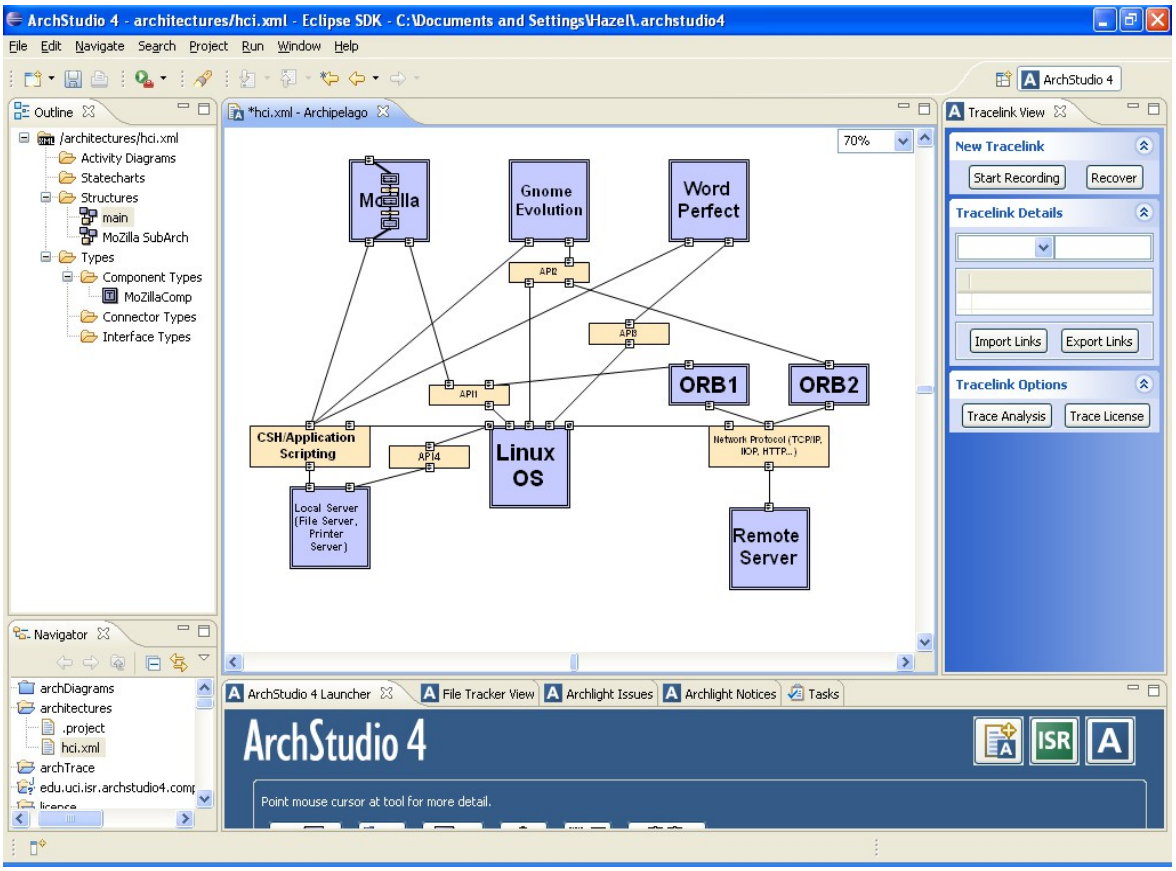
	Actor	Modality	Action	Object	License (optional)
Abstract Right	*Licensee or Licensor*	*May or Need Not*	The set of actions is large, comprising whatever actions the licenses in question utilize	*Any Under This License*	*This License or Object's License*
				Any Source Under This License	
				Any Component Under This License	
Concrete Right				Concrete Object	Concrete License
Concrete Obligation					
Abstract Obligation		*Must or Must Not*		*Right's Object*	*Concrete License or Right's License*
				All Sources Of Right's Object	
				X Scope Sources	
				X Scope Components	

License inference scheme

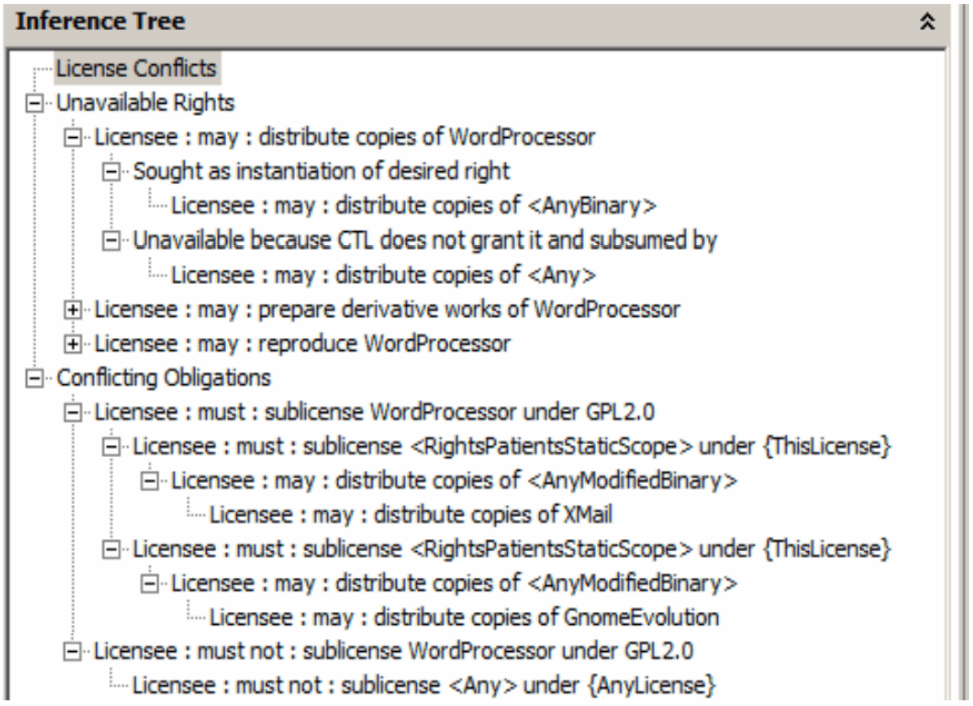


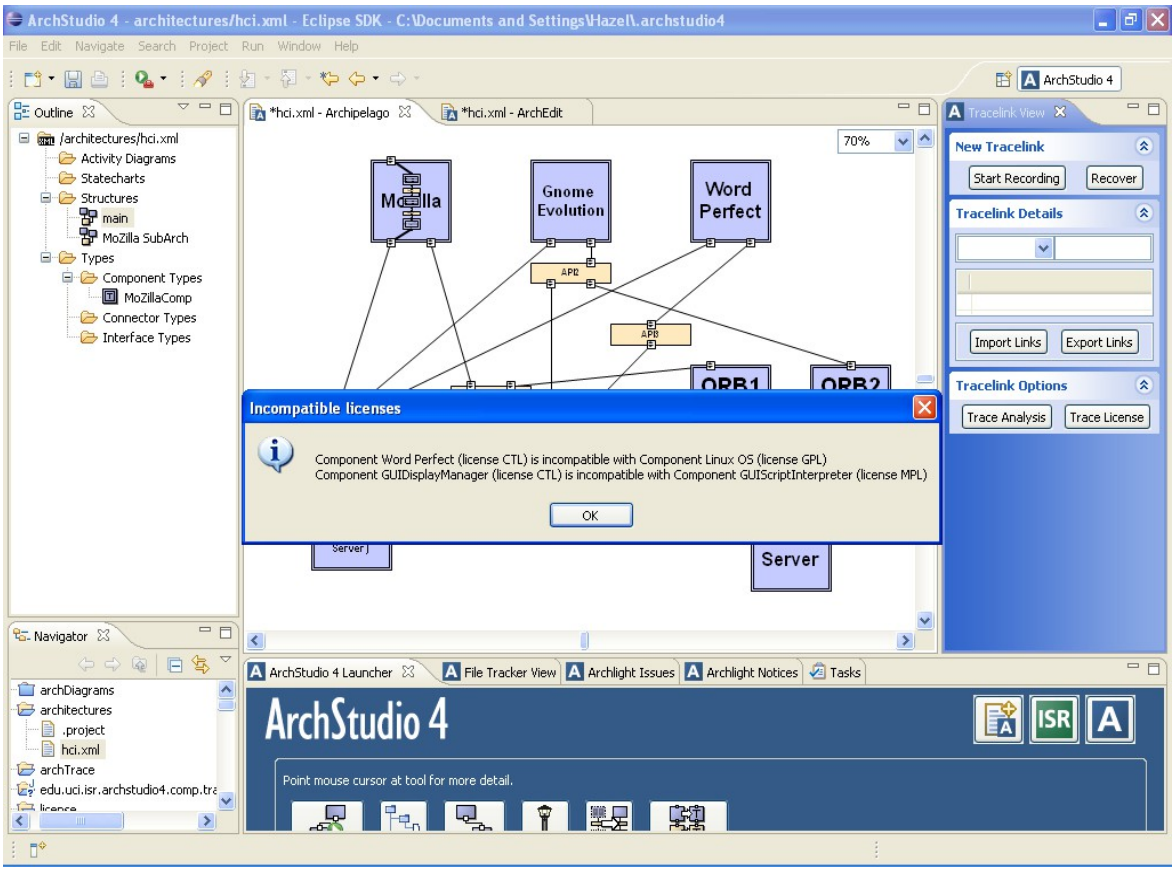
Software IP/Security license analysis

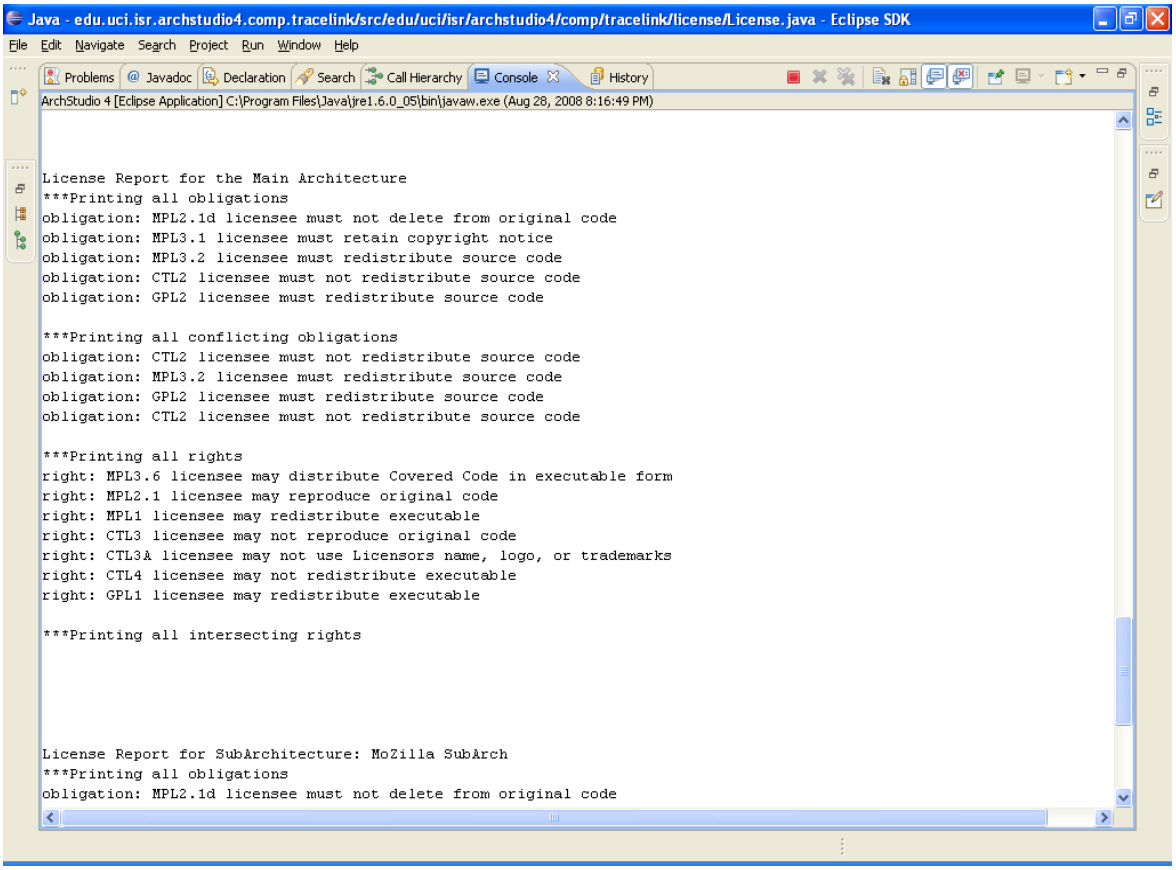
- License types:
  - Strongly reciprocal, weakly reciprocal, academic, Terms of Service, Proprietary
- Propagation of reciprocal obligations
- Conflicting obligations
- Calculating obligations and rights



Reasoning structure during analysis







Challenges of securing open architecture (OA) C2 systems

Security challenges

- Security threats to software systems are increasingly multi-modal and distributed across system components.
- Physically isolated systems are vulnerable to external security attacks.
- What makes an OA C2 system secure changes over time, as new threats emerge and systems evolve.
- Need an approach *to continuously assure the security of evolving OA C2 systems* that is practical, scalable, robust, tractable, and adaptable.

Software systems/components evolve: what to do about security?

- Individual components evolve via revisions (e.g., security patches)
- Individual components are updated with functionally enhanced versions;
- Individual components are replaced by alternative components;
- Component interfaces evolve;
- System architecture and configuration evolve;
- System functional and security requirements evolve;
- System security policies, mechanisms, security components, and system configuration parameter settings also change over time.

Current security approaches

- Mandatory access control lists, firewalls;
- Multi-level security;
- Authentication (including certificate authority and passwords);
- Cryptographic support (including public key certificates);
- Encapsulation (including virtualization), hardware confinement (memory, storage, and external device isolation), and type enforcement capabilities;
- Secure programming practices;
- Data content or control signal flow logging/auditing;
- Honey-pots, traps, sink-holes;
- Security technical information guides (STIGs) for configuring the security parameters for applications and operating systems;
- Functionally equivalent but diverse multi-variant software executables.
- Software component security assurance processes.

Current approaches to software system security do not address the challenges of continuously evolving OA C2 systems emerging within agile, adaptive software ecosystems!

Case Study: Securing the development and evolution of an OA C2 system within an agile, adaptive software ecosystem

Carefully specifying security policy obligations and rights

- The obligation for a user to verify his/her authority to see compartment T, by password or other specified authentication process
- The obligation for all components connected to specified component C to grant it the capability to read and update data in compartment T
- The obligation to reconfigure a system in response to detected threats, when given the right to select and include different component versions, or executable component variants.
- The right to read and update data in compartment T using the licensed component
- The right to add, update, replace specified component D in a specified configuration
- The right to add, update, or remove a security mechanism
- The right to update security policy L.

Software product lines?

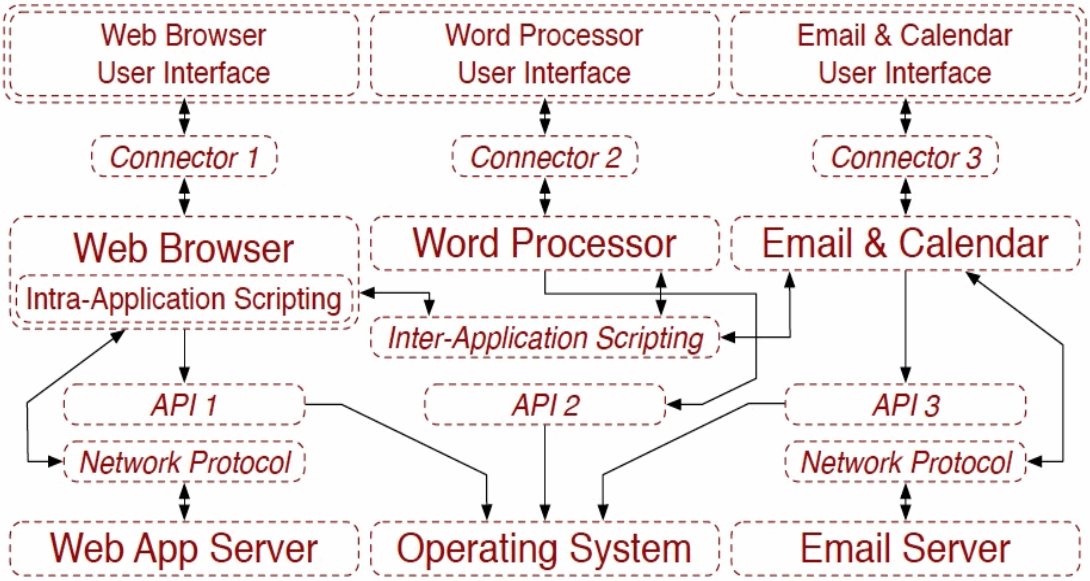
- When functionally similar software components, connectors, or configurations exist,
- Such that equivalent alternatives, versions, or variants may be substituted for one another, then
- We have a strong relationship among these OA system elements that denotes a *software product line*.
- Software product lines for OA systems enable support from agile, adaptive software (component) ecosystems

- Reed, H., Benito, P., Collens, J. and Stein, F. (2012). Supporting Agile C2 with an Agile and Adaptive IT Ecosystem, *Proc. 17Th Intern. Command and Control Research and Technology Symposium* (ICCRTS), Paper-044, Fairfax, VA, June 2012.

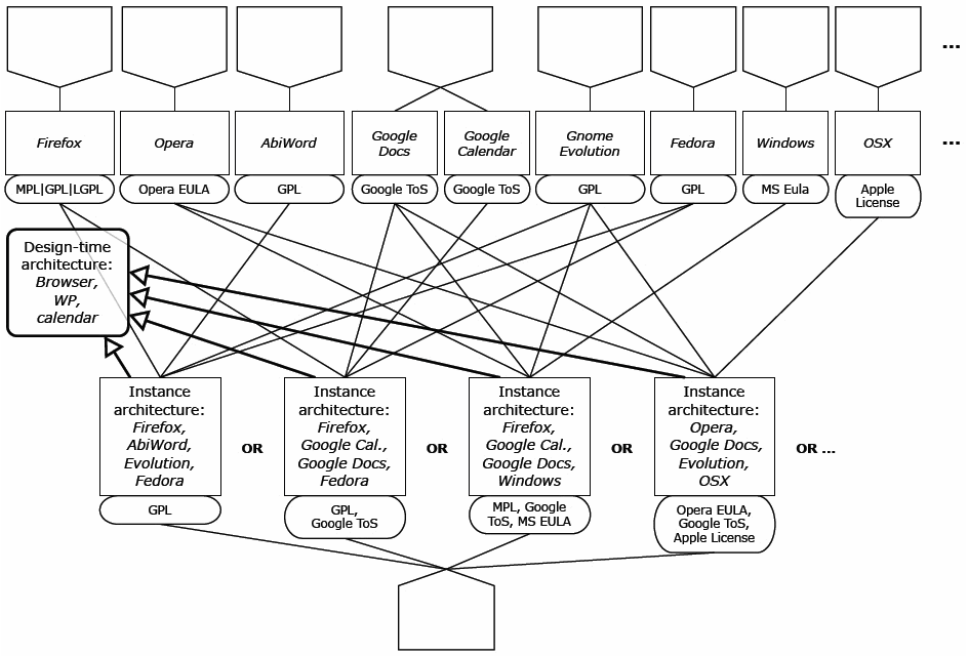
Security challenges

- Security threats to software systems are increasingly multi-modal and distributed across system components.
- Physically isolated systems are vulnerable to external security attacks.
- What makes an OA C2 system secure changes over time, as new threats emerge and systems evolve.
- Need an approach *to continuously assure the security of evolving OA C2 systems* that is practical, scalable, robust, tractable, and adaptable.

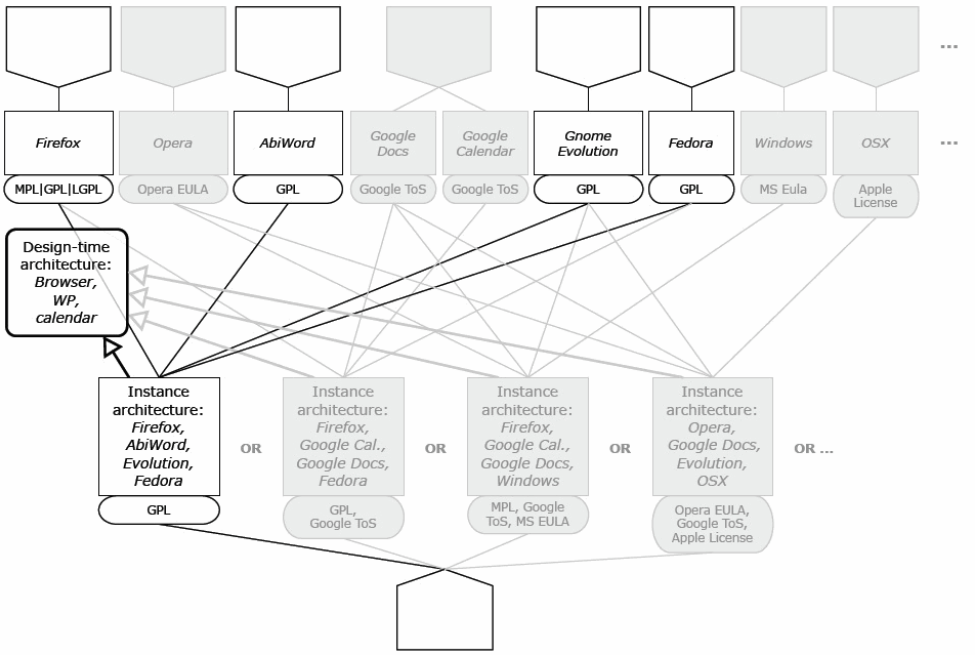
Design-time view of an OA system



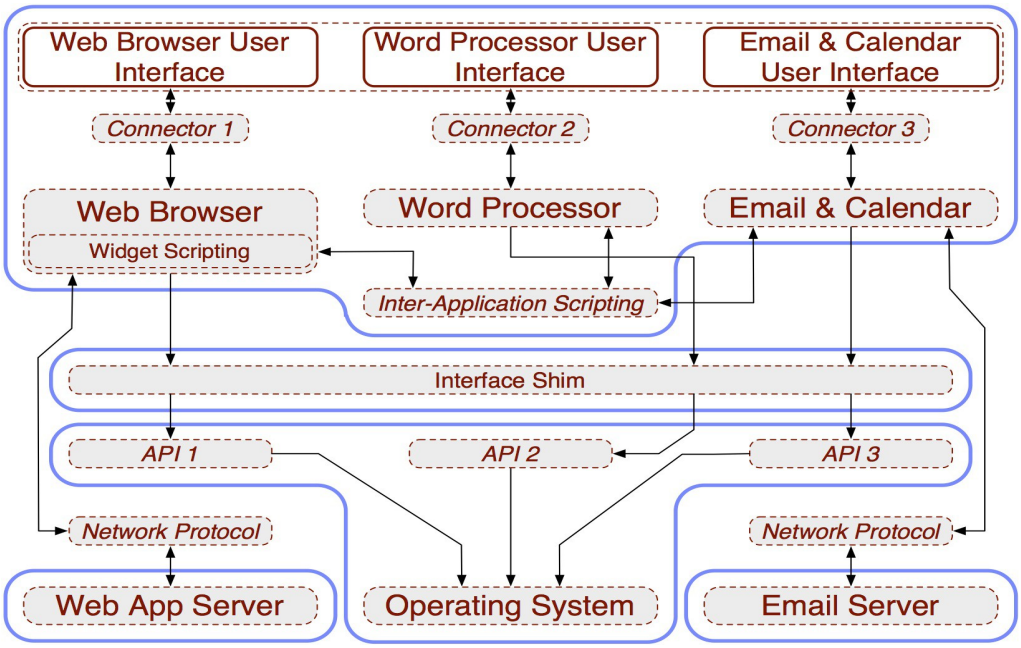
Software product line of *functionally similar* OA system alternatives



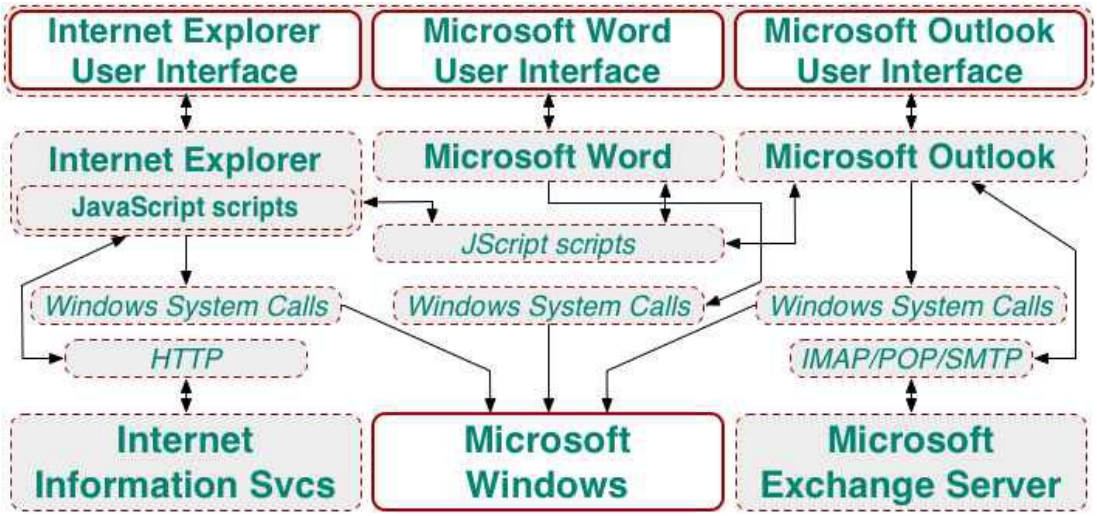
Product line selection of one alternative system configuration



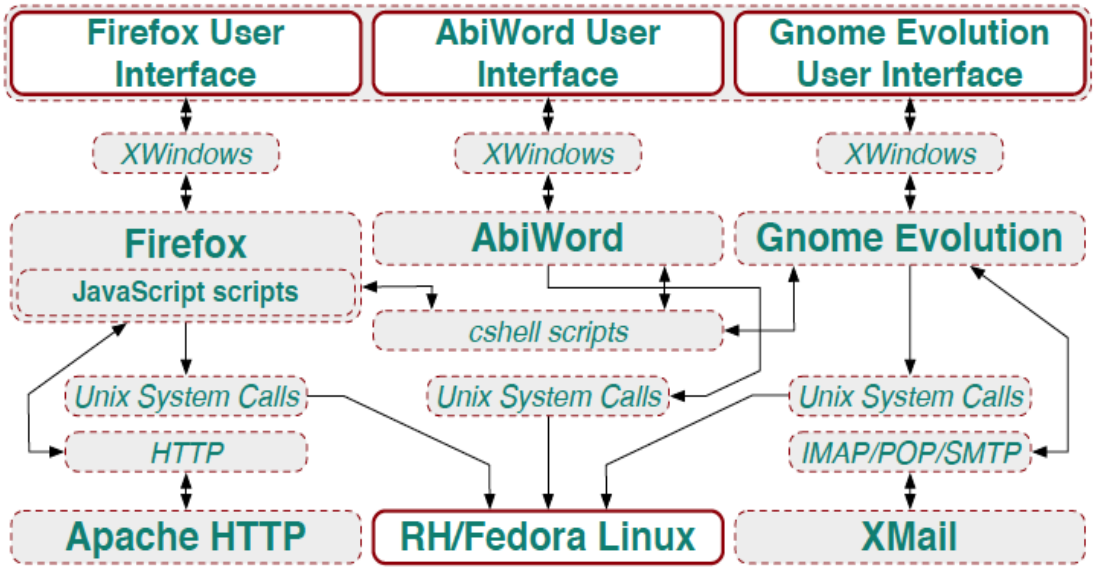
A security requirement encapsulating the *design-time* configuration via multiple virtual machine containers



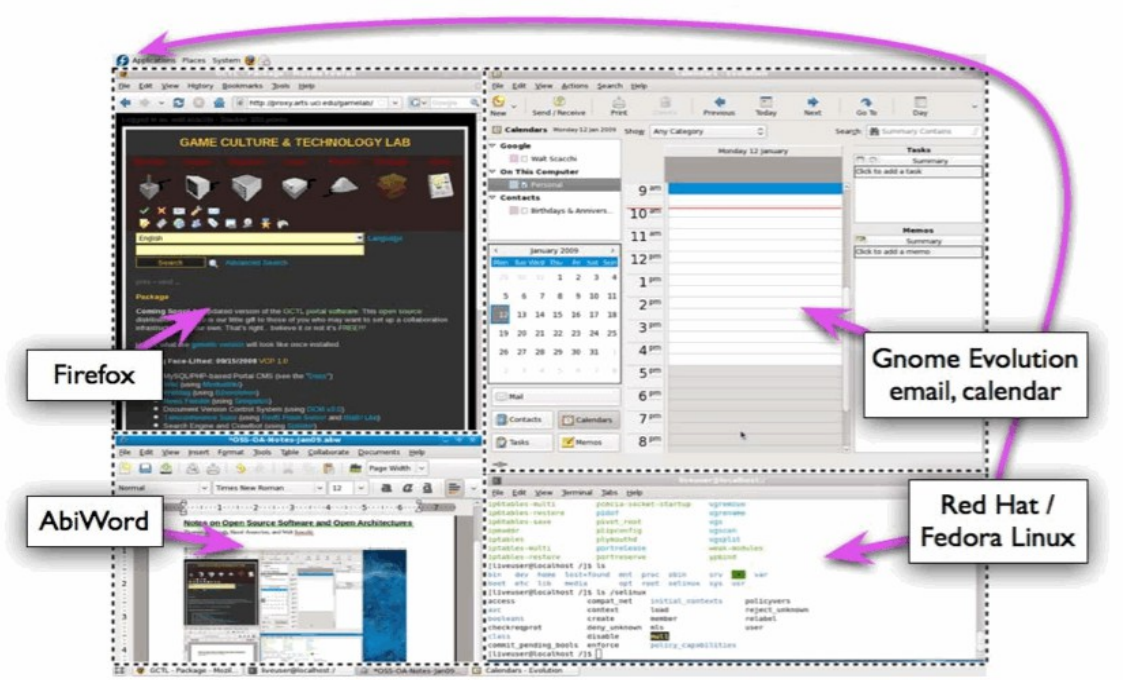
Build-time view of OA design selecting *proprietary* product family alternatives



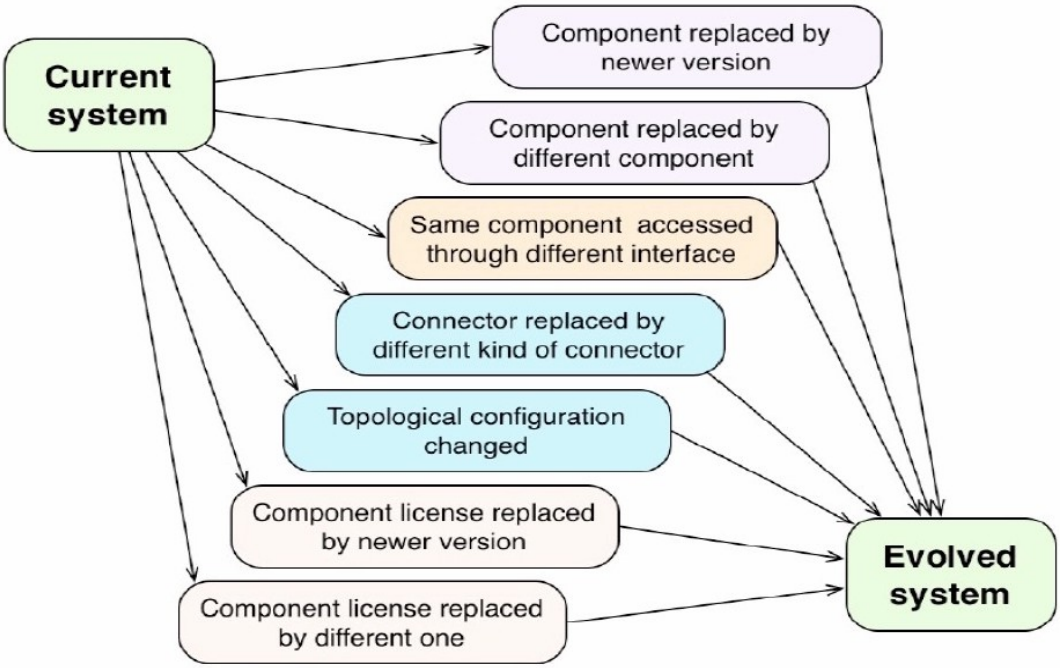
Build-time view of OA design selecting *OSS* product family alternatives



Run-time deployment view of OA system family member configuration



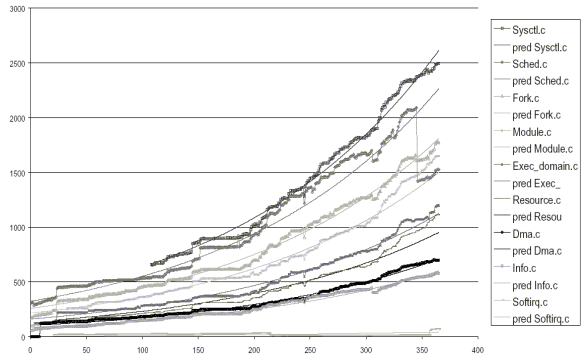
Evolution-time software changes

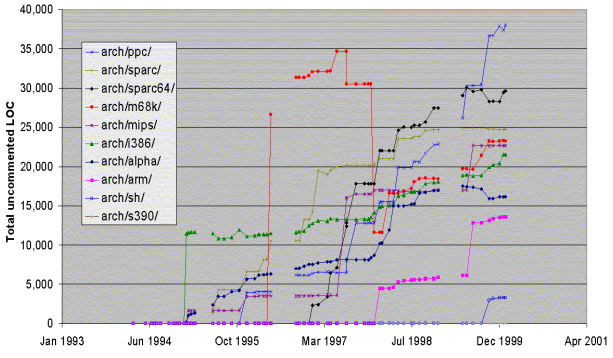


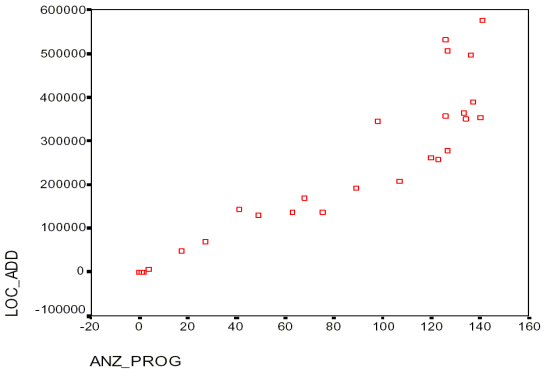
Evolutionary redevelopment, reinvention, and redistribution

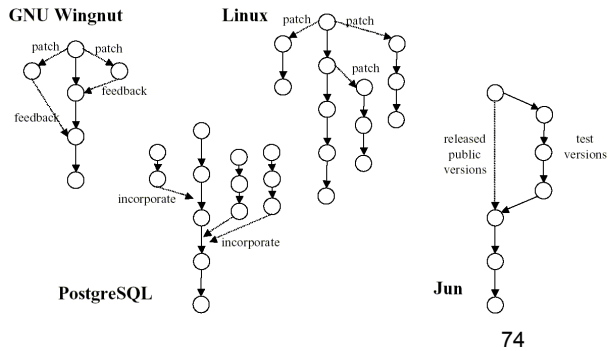
- Overall evolutionary dynamic of many OSSD projects is *reinvention and redevelopment*
  - Reinvention enables continuous improvement and collective learning
- OSS evolve through minor mutations
  - Expressed, recombined, redistributed via incremental releases
- OSS systems *co-evolve* with their development community
  - Success of one depends on the success of the other

Institute for Software Research, UCI



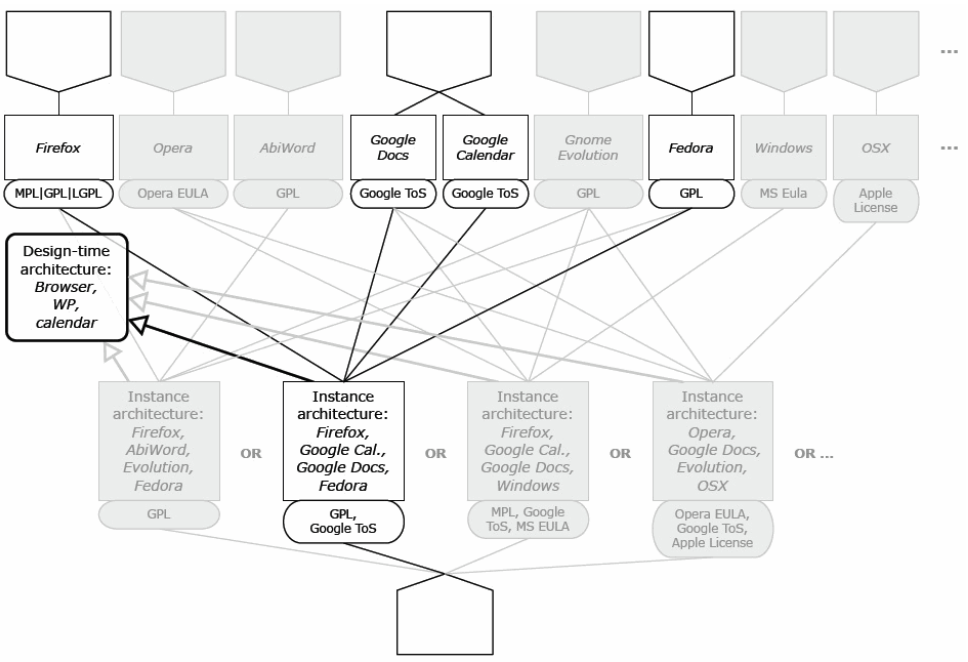




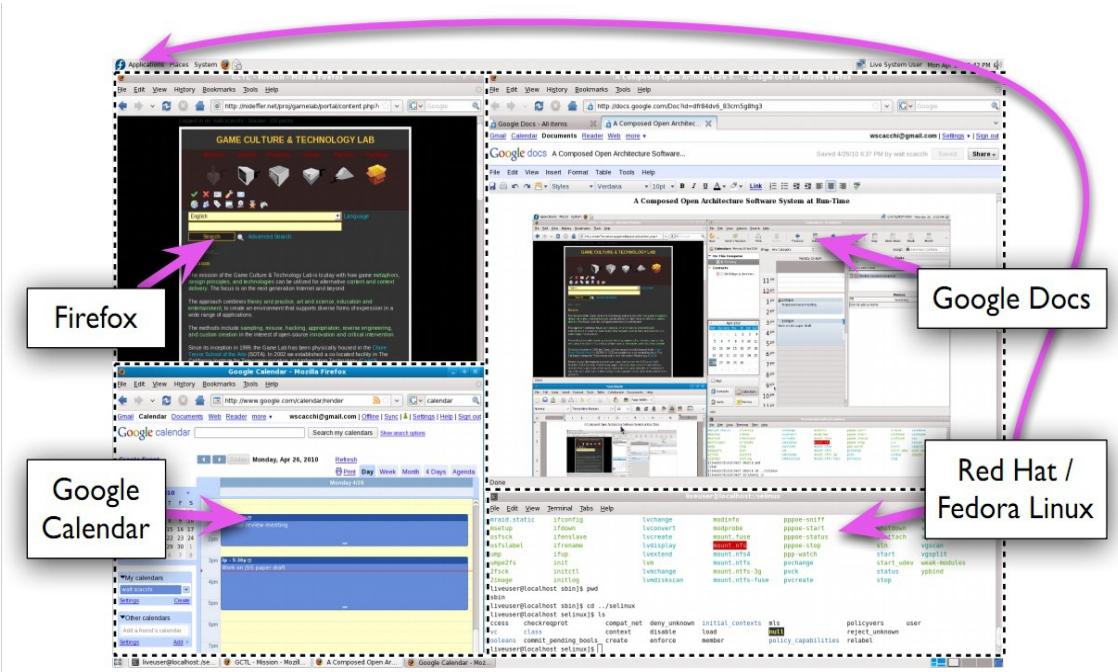


74

Product line selection of different functionally similar alternative



Evolved run-time deployment view of a *functionally similar* alternative OA system configuration



Emerging Transformations with OSS and OA systems

New paths for OA C2 development and evolution using widgets/apps acquired from online App Store

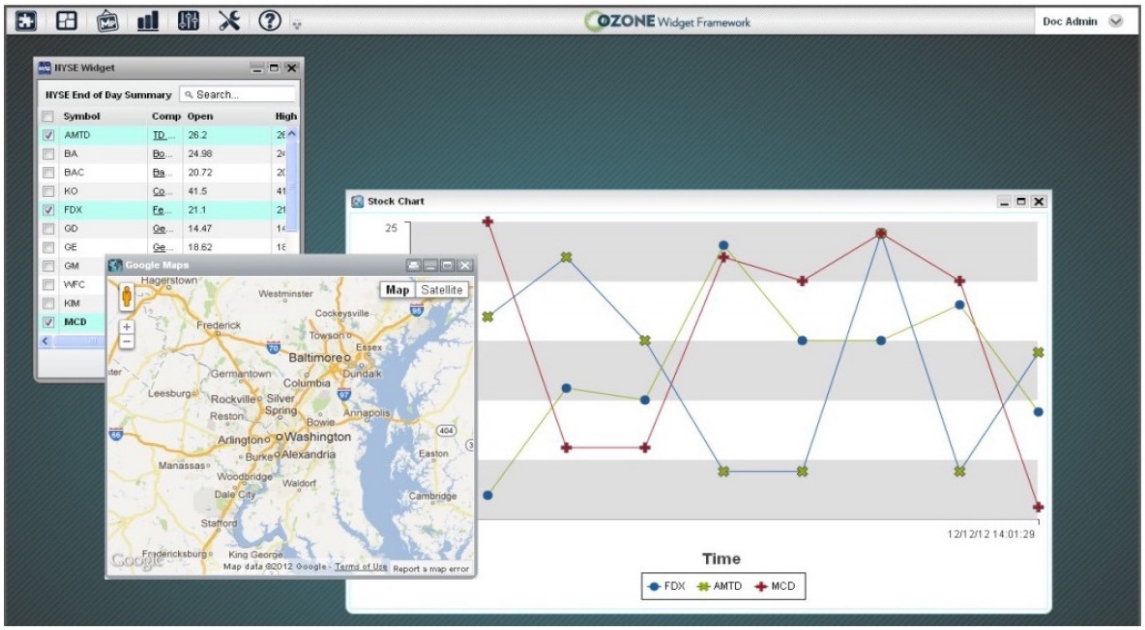
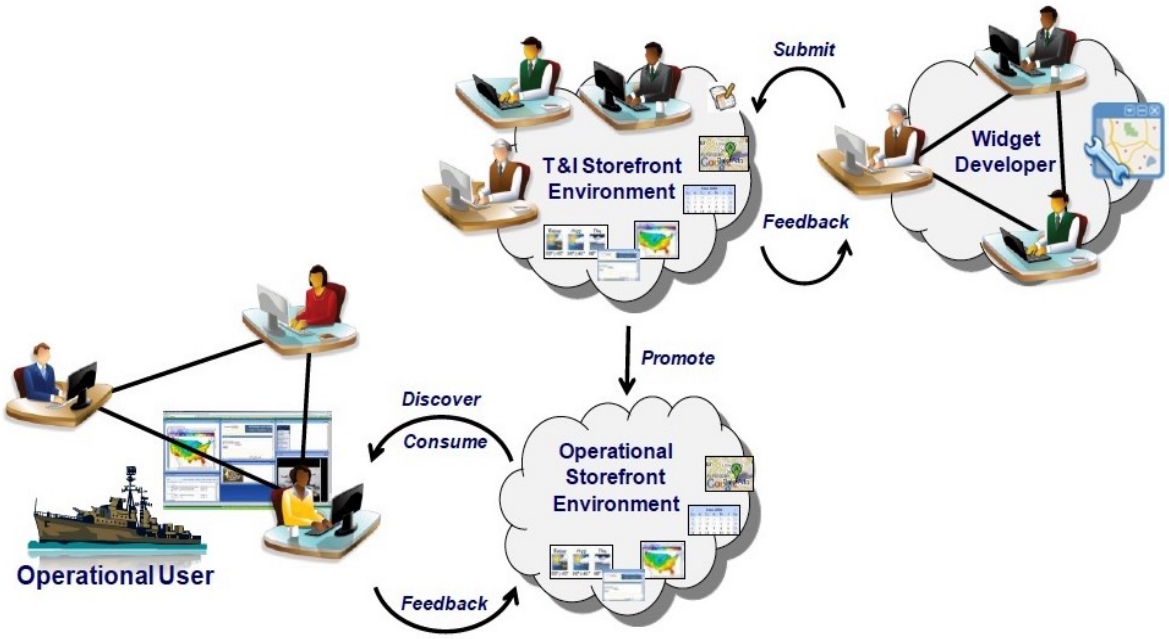
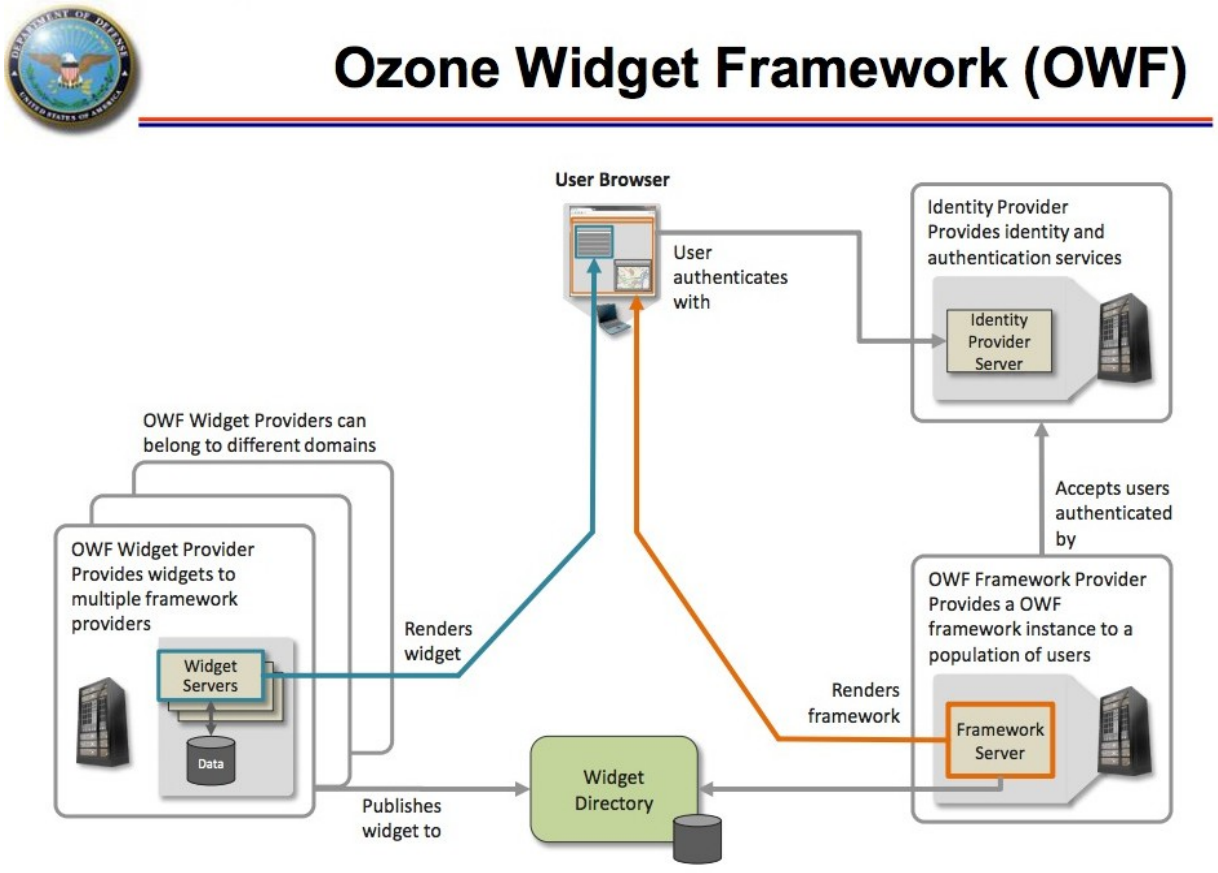
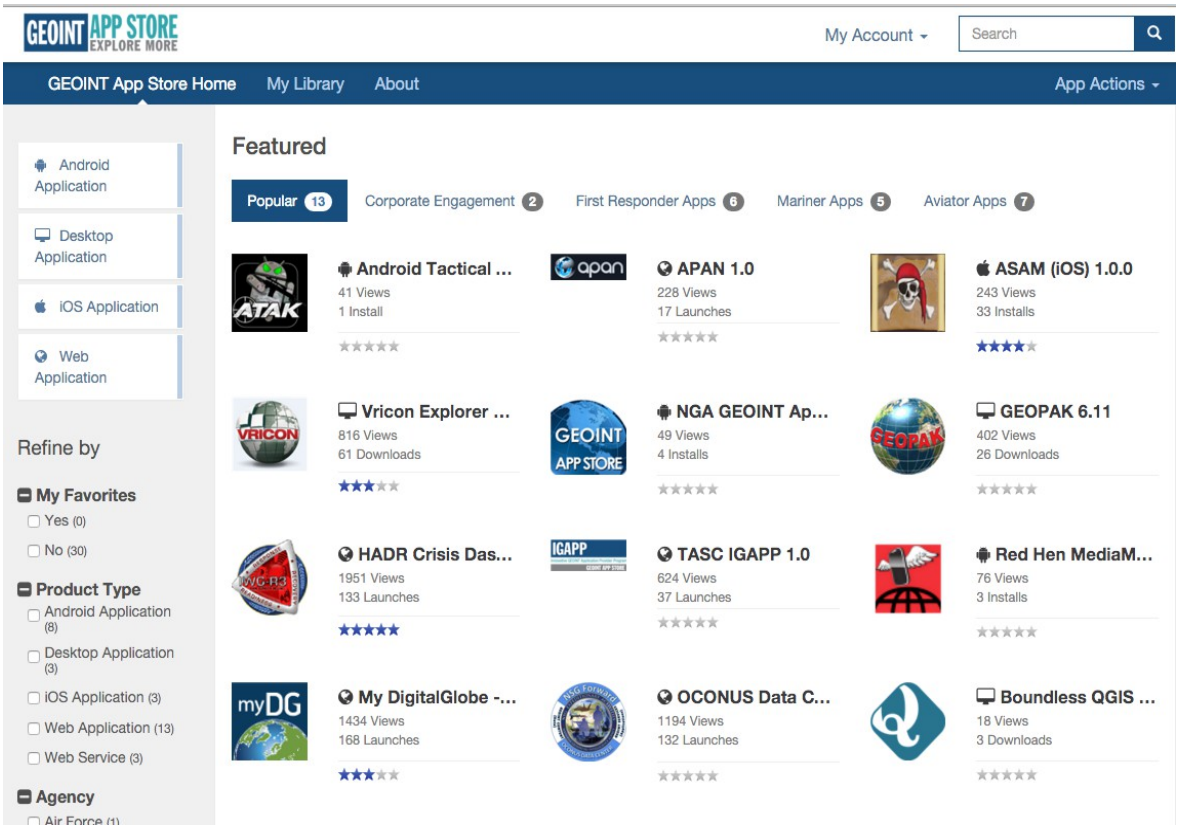


Figure 1: PEO C4I Storefront Operational Concept

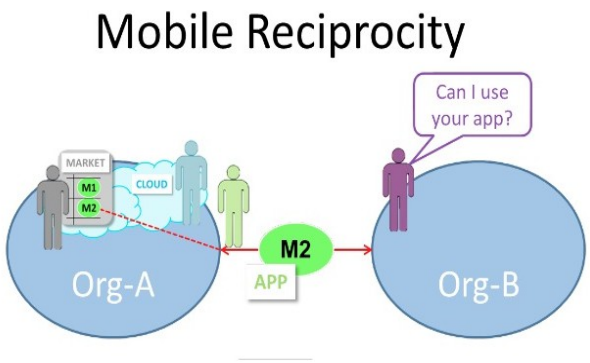


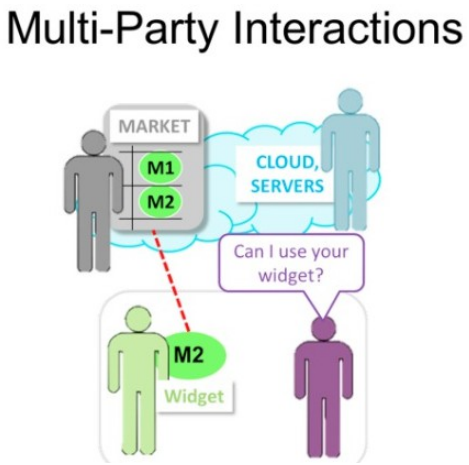
Source: George, A. Galdorisi, G, Morris, M. O"Neil, M. (2014). DoD Application Store: Enabling C2 Agility?, *Proc. 19th Intern. Command and Control Research & Technology Symposium*, Alexandria, VA.





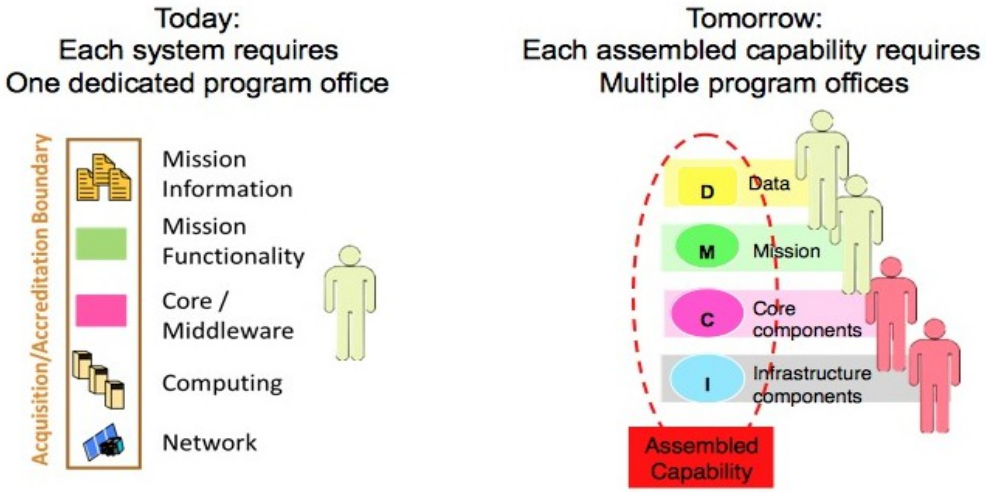
Transforming to multi-party acquisition of software elements within OA ecosystems





Customer/end-user organizations now looking for ways to reduce acquisition cost and effort through *shared development/use of common OA software system components* (apps, widgets).

Transforming to multi-party acquisition of software elements within OA ecosystems



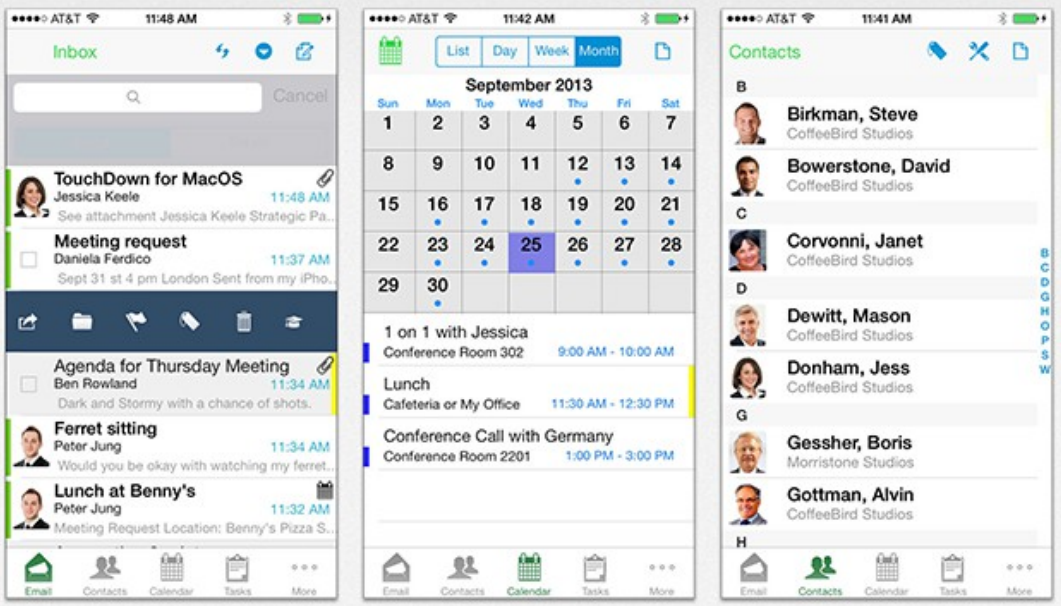
New Core Competencies: Provide components, Assemble capabilities

Customer/end-user organizations now looking for ways to reduce acquisition cost and effort through *shared development/use of assembled capabilities* for C3CB systems

Shared development of Apps and Widgets as OA system components

Ozone Widget Framework for Web PCs and Mobile Devices

Commercial Mobile Apps also being used (middleware services, not shown)



Enterprise-to-Mobile Middleware *IP Licenses*
(for the NitroDesk *Touchdown* product in 2014)

* LGPL 2.1
* Ical4j from Ben Fortuna
* Public Domain Declaration
* Apache 2
* The Legion of the Bouncy Castle
* Creative Commons BY
* Sony Mobile
* Jesse Anderson
* OpenSSL
* Apple Non-Exclusive
* SQLite
* Microsoft Public License

Growing diversity of challenges in cybersecurity

- Scacchi, W. and Alspaugh, T. (2012) Addressing Challenges in the Acquisition of Secure Software Systems with Open Architectures, *Proc. 9th Acquisition Research Symposium*, Vol. 1, 165-184, Naval Postgraduate School, Monterey, CA.
- Scacchi, W. and Alspaugh, T. (2013a). Processes in Securing Open Architecture Software Systems, *Proc. 2013 Intern. Conf. Software and System Processes*, San Francisco, CA, May 2013.
- Scacchi, W. and Alspaugh, T.A. (2013b). Streamlining the Process of Acquiring Secure Open Architecture Software Systems, *Proc. 10th Annual Acquisition Research Symposium*, Monterey, CA, 608-623, May 2013.
- Scacchi, W. and Alspaugh, T.A. (2013c). Challenges in the Development and Evolution of Secure Open Architecture Command and Control Systems, *Proc. 18th Intern. Command and Control Research and Technology Symposium*, Paper-098, Alexandria, VA, June 2013.

Shared development of Apps and Widgets as OA system components: *Cybersecurity?*

Ozone Widgets supporting "Bring Your Own Devices" (BYOD)?

New business models for acquisition of OA Web/mobile software components

- Franchising
- Enterprise licensing
- Metered usage
- Advertising supported
- Subscription
- Free component, paid service fees
- Federated reciprocity for shared development
- Collaborative buying
- Donation
- Sponsorship
- Government open source software (GOSS)
- and others

Emerging challenges in achieving *Better Buying Power* via OA software systems

- Program managers/staff *may not understand* how software IP licenses affect OA system design, and vice-versa.
- Software IP and cybersecurity obligations and rights propagate across system development, deployment, and evolution activities *in ways not well understood* by system developers, integrators, end-users, or acquisition managers.

Emerging challenges in achieving *Better Buying Power* via OA software systems

- *Failure to understand* software IP and cybersecurity obligations and rights propagation can reduce DoD buying power, increase software life cycle costs, and reduce competition.
- DoD, other Government agencies, and large enterprises *would financially and administratively benefit* from engaging the OSSD and deployment of an *automated software obligations and rights management system* for the software asset management (acquisition) workforce.

New practices to realize cost-effective acquisition of OA software systems

- Need to R&D ***worked examples*** of reference OA system models, and component evolution alternatives.
- Need ***open source models of*** app/widget security assurance ***processes and*** reusable cybersecurity ***requirements.***
- Need precise ***domain-specific languages*** (DSLs) and ***automated analysis tools*** for continuously assessing and continuously improving cybersecurity and IP requirements for OA C2 systems composed from apps/widgets.

Conclusions

- Our research identifies how new OSS component technologies, IP and security requirements, OSS development practices, and new business models interact to drive-down or drive-up C2 system costs.
- New technical risks for component-based OA software systems can dilute the cost-effectiveness of OSSD efforts.
- Need R&D leading to automated systems that can model and continuously analyze OA system IP licenses and cybersecurity requirements
  - Empower OA C2 system development workforce
  - Identify and manage cost-effectiveness trade-offs

Acknowledgements

Research collaborators

- Mark Ackerman, UMichigan, Ann Arbor; Kevin Crowston, Syracuse U; Les Gasser, UIllinois, Urbana-Champaign; Chris Jensen, Google; Greg Madey, Notre Dame U; John Noll, LERO; Megan Squire, Elon U; and others.
- Thomas Alspaugh, Hazel Asuncion (UWashington-Bothwell), Margaret Elliott, and others at the UCI ISR.
- DASD(A)-C3CB *Assembled Capabilities Working Group*.

Acknowledgements

Funding support ***(No endorsement, review, or approval implied)***.

- National Science Foundation: #0083075, #0205679, #0205724, #0350754, #0534771, #0749353, #0808783, and #1256593.
- Naval Postgraduate School
  - *Acquisition Research Program* (2007-2015+)
    - N00244-1-15-0010 (2015-2016)
  - *Center for the Edge Research Program* (2010-2012).
- Computing Community Consortium (2009-2010).

Thank you!

INSTITUTE *for* SOFTWARE RESEARCH
UNIVERSITY *of* CALIFORNIA • IRVINE

THIS PAGE LEFT INTENTIONALLY BLANK

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL
555 DYER ROAD, INGERSOLL HALL
MONTEREY, CA 93943

www.acquisitionresearch.net